Пояснювальна записка
до дипломного проекту
на тему: «Централізована система візуалізації критичних повідомлень засобів моніторингу »

Київ – 2019 рік

					ІА51.300БАК.005 ПЗ		
Зм.	*Арк.*	*№ докум*	*Підпис*	*Дата*			
Розроб.		*Ягодзінський К.А.*			Централізована система візуалізації критичних повідомлень засобів	*Літ.*	*Аркуш* 1 / *Аркушів*
Перевір.		*Дорогий Я.Ю.*					
Н. контр.						*НТУУ "КПІ ім. І. Сікорського", ФІОТ, гр. ІА-51*	
Затверд.							

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AS	(Autonomous system) Автономна система
IP	(Internet Protocol) Ідентифікатор
HA	(High availability) Висока доступність
FT	(Fault Tolerance) відмовостійкість
ОС	Операційна система
ISP	(Internet Service Provider) Постачальник послуг Інтернет
SDK	(software Development Kit) Набір засобів розробки

ВСТУП

Ріст та розвиток інформаційних ресурсів, зобов'язує реалізацію рішень для моніторингу цих ресурсів. У сучасному світі більшість проектів розгортаються за допомогою хмарних технологій, таких як AWS (Amazon Web Service), Microsoft Azure, GCP (Google Cloud Platform), IBM Bluemix, Alibaba Cloud, Hetzner Cloud, Heroku, Huawei Cloud, DigitalOcean. Зазвичай у різних провайдерів хмарних технологій присутні власні рішень для реалізації моніторингу використовуємих сервісів, як приклад Amazon Web Services пропонує AWS Cloudwatch для збирання метрик сервісів, Google Cloud Platform використовує для таких цілей Google Stack Driver, Microsoft Azure використовує Azure Monitor. Проблема таких систем моніторингу полягає у тому, що зазвичай дані системи моніторингу збирають метрики саме сервісів, які надають провайдери, наприклад при розгортанні програмного забезпечення, яке використовує базу даних на базі Amazon Web Services знадобиться використання таких сервісів, як AWS EC2 та AWS RDS.

Приведені проблеми стосуються не тільки системи від Amazon Web Services, а також в тому чи іншому вигляді можуть бути наявні і в інших провайдерів хмарних послуг. Проте навіть враховуючи недоліки таких систем моніторингу, їхнє використання є доцільним, оскільки вони прості в налаштуванні, мають детальну документацію та не потребують накладних витрат для реалізації Fault Tolerance (відмовостійкості) та High Availability (доступності), оскільки відповідальність за їх підтримку бере на себе сам провайдер хмарних послуг.

Тому для побудови повноцінного моніторингу в контексті хмарних технологій часто використовуються декілька систем моніторингу – одна стандартна для обраного хмарного провайдера, а інша – самостійно розгорнута. Такий підхід дає змогу збирати якомога більше метрик, які

стосуються як інфраструктури, так і самого програмного забезпечення, що забезпечує більш стабільну роботу програмного забезпечення в цілому.

Проте при такому підході виникає проблема агрегації критичних повідомлень різних систем моніторингу. Зазвичай реагуванням на критичні повідомлені, які можуть впливати на стабільність роботи програмного забезпечення, займається окрема група інженерів. Наявність більше ніж одної системи моніторингу суттєво впливає на час дослідження причин виникнення критичного повідомлення, оскільки це зменшує прозорість інфраструктури як такої.

Метою проекту є аналіз популярних систем моніторингу, та розробка програмного забезпечення, як виконувала інтеграцію з різними системами моніторингу та відала дані про критичні повідомлення від них.

Для досягнення поставленої мети необхідно:

1) дослідити існуючі системи моніторингу, принципи їх дії, архітектури методи взаємодії з ними, наявність http API для отримання даних про критині повідомлення. Також необхідно дослідити особливості їх розгортання у різних середовищах та експлотації.
2) дослідження та реалізація основних принципів проектування, таких як відмовостійкість, наявність резервних копій, висока доступність, контейнеризація (з використанням технології docker).
3) дослідження інструментів візуалізації даних, які можна використати як інтерфейс для відображення критичних повідомлень різних систем моніторингу, інтеграція реалізованого програмного забезпечення з ним.
4) дослідження та використання технології для реалізації програмного забезпечення, яке буде інтегруватися с різними системами моніторингу.

5) дослідження та використання принців розгортання програмного забезпечення і систем моніторингу локально та у хмарному середовищі.

РОЗДІЛ 1. РІЗНОВИДИ СИСТЕМ МОНІТОРИНГУ

1.1. Загальна характеристика систем моніторингу

На даний момент існує багато систем моніторингу, кожна з яких відрізняється реалізацією та використовуються у певних умовах в залежності від середовища використання, архітектури програмного забезпечення тощо. Наприклад часто використовуються такі системи моніторингу:

1) Cloudwatch;
2) StackDriver;
3) Azure Monitor;
4) Nagios;
5) Prometheus;
6) Datadog;
7) Icinga2;
8) Zabbix.

При розгортанні інфраструктури програмного забезпечення за допомогою хмарних провайдерів найбільш очевидним вибором є саме їх внутрішні системи моніторингу. Вони не потребують зусиль для розгортання та достатньо прості у конфігурації. Найбільш поширеним на даний момент є Cloudwatch оскільки AWS є найпопулярнішим хмарним провайдером на даний момент. Amazon Web Services займає приблизно більшу частину ринку хмарних платформ у порівняні с іншими, такими як Google Cloud Platform, Microsoft Azure тощо.

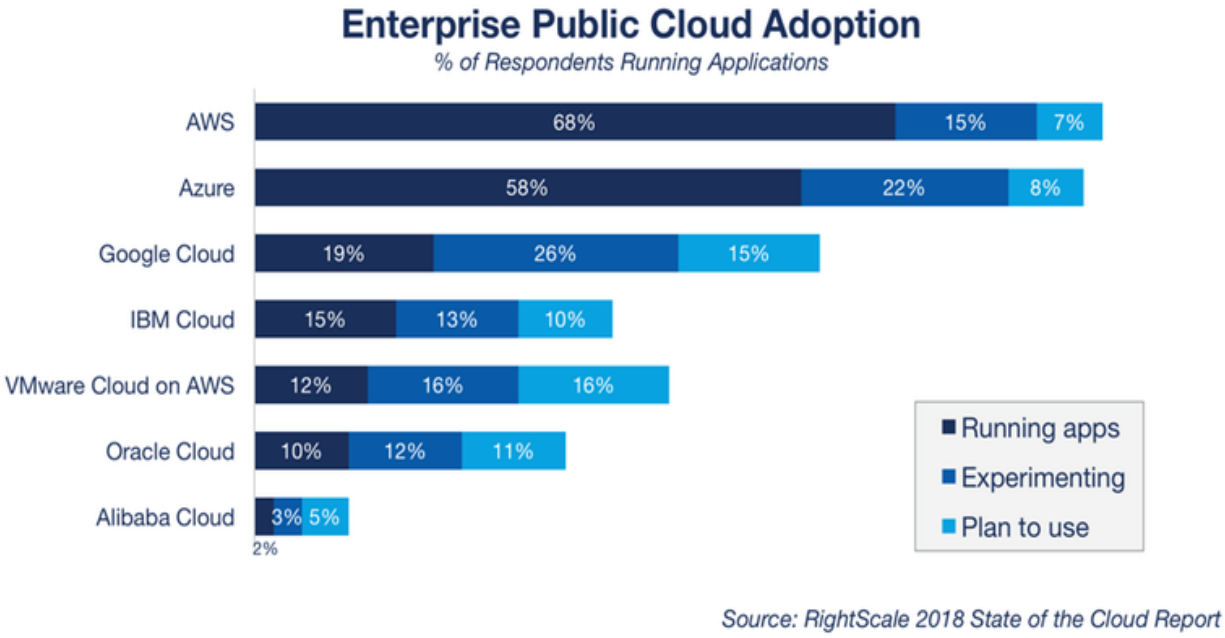

Рисунок 1.1. – Порівняння хмарних провайдерів

Серед систем моніторингу, які не є стандартними для хмарних платформ, вибір набагато більший, а тому розподілення не таке однозначне. Одними з найпопулярніших є Nagios та Zabbix, оскільки вони існують достатньо давно та своєрідним стандартом. Багато інженерів вже мають необхідні знання для їх розгортання та конфігурації, тому їх використання є достатньо зручним. Також достатньо популярними є системи моніторингу, які не є частиною хмарних платформ, але розповсюджуються по моделі SaaS, тобто у якості сервісу, такі як Datadog. Зі збільшенням популярності мікросервісної архітектури більшого ужитку набирає система моніторингу Prometheus, розроблена для таких цілей компанією SoundCloud, спочатку як їх внутрішня система моніторингу, але згодом отримала більш широке застосування.

1.2. Cloudwatch.

1.2.1 Загальний опис

Amazon CloudWatch є компонентом Amazon Web Services (AWS), який забезпечує моніторинг ресурсів AWS і клієнтських додатків, що працюють на інфраструктурі Amazon.

CloudWatch забезпечує моніторинг ресурсів AWS у реальному часі, таких як Amazon EC2, Amazon EBS (Elastic Block Store), AWS ELB та екземпляри бази даних Amazon RDS. Програма автоматично надає метрики для використання процесора, затримки та кількості запитів; користувачі можуть також передбачити додаткові показники, які підлягають моніторингу, такі як використання пам'яті, обсяги транзакцій або частота помилок.

Користувачі можуть отримати доступ до функцій CloudWatch через API, засоби командного рядка, один з AWS SDK (комплекти розробки програмного забезпечення) або консоль керування AWS. Інтерфейс CloudWatch надає поточну статистику, яку можна переглянути в графічному форматі.

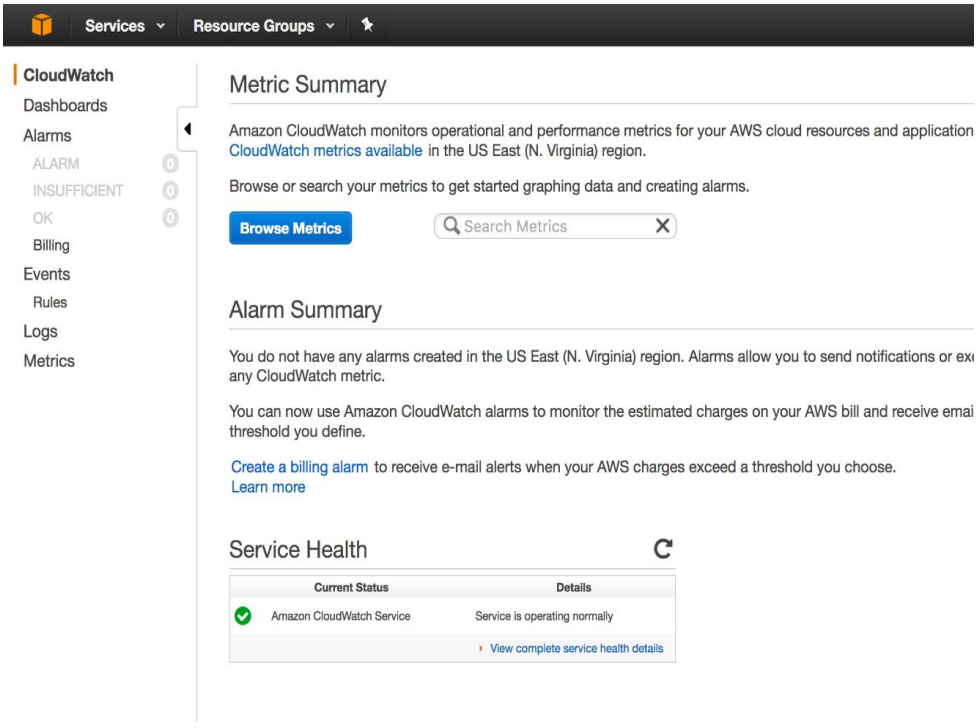

Рисунок 1.2. – Веб-консоль CloudWatch

Користувачі можуть встановлювати сповіщення (звані "нагадування"), які надсилаються, коли щось, що підлягає контролю, перевищує заданий поріг. Програма також може виявляти та вимикати невикористані або невикористані екземпляри EC2.

Основні можливості Cloudwatch:

Моніторинг сервісів Amazon Web Services. Метрики доступні для всіх сервісів AWS, таких як AWS EC2, AWS RDS, AWS ElasticCache, AWS S3 тощо.

Збір логів програмного забезпечення та їх відображення за допомогою консолі керування AWS.

Створення власних метрик, наприклад, для моніторингу програмного забезпечення.

«Події», які ввідображають зміни, які відбуваються у інфраструктурі AWS. Можуть бути використані, коли потрібно виконати певну дію, коли в інфраструктурі з’являється якась подія. У якості обробників таких подій використовуються сервіс AWS Lambda.

Створення так “Event buses”, які дають можливість відправляти та отримувати події з різних AWS аккаунтів.

Відслідковування поточних грошових витрат на використання сервісів AWS та створення відповідних попереджень, побудова статистики.

Створення користувацьких дашбоардів для візуалізації метрик, які можуть бути використані для швидкого доступу до важливих метрик, відображення їх на великих моніторах тощо.

Створення алертів, які допомагають відслідковувати стан важливих сервісів та отримувати повідомлення у випадку, якщо певна метрика або сервіс знаходиться у критичному стані. Можлива інтеграція с різними месседжерами, такими як slack, telegram, надсилання

повідомлень через електрону пошту, інтеграція з платформами управління інцидентами – Victorops, Opsgenie, Pagerduty тощо.

1.2.2. Архітектура Cloudwatch

Загальна схема архітектури CloudWatch вказана на рис.1.3.:

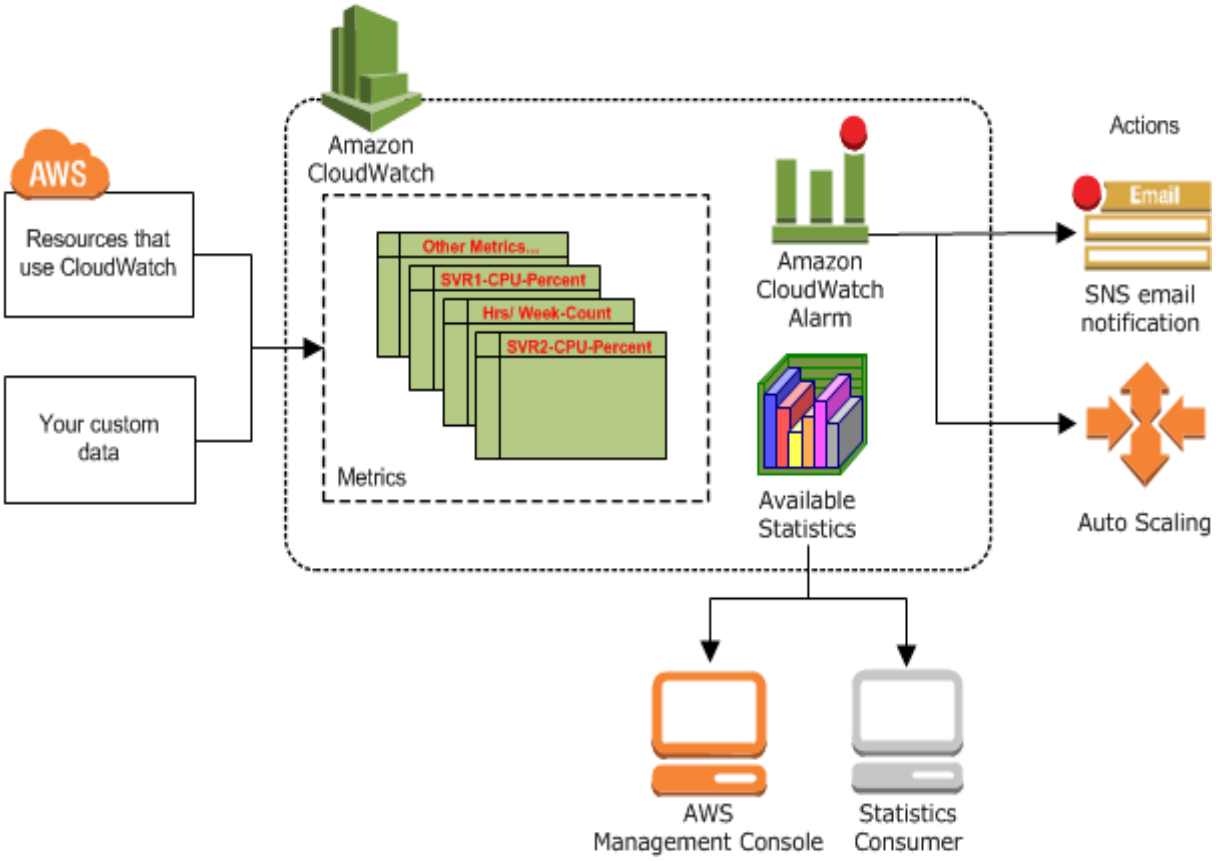

Рисунок 1.3. – Загальна схема архітектури Cloudwatch.

На вхід до системи моніторингу Cloudwatch подаються метрики. Вони можуть бути згенеровані або ресурсами AWS, які використовують Cloudwatch або це можуть бути довільні данні, створенні іншим джерелом. Усі вони потрапляють до Cloudwatch та зберігаються там. Дані мають ієрархічну структуру. Основні терміни пов'язані зі зберіганням даних:

1) Namespace (Простір імен);
2) Метрика;

3) Dimensions (вимір);
4) Статистика;
5) Агрегація;
6) Процентилі;

Простір імен є чимось на кшталт контейнеру для метрик, отриманих від зовнішніх сервісів. Метрики с різних просторів імен є ізольованими один від одного, тому якщо декілька сервісів надсилають однакову метрику до система моніторингу, але використовують різні простори – це не вплине на агрегацію та відображення даних, тобто різні сервіси не будуть помилково впливати один на одного.

В Cloudwatch немає стандартного простору імен, тобто при надсиланні даних до системи моніторингу обов'язково вказувати простір імен, до якого відноситься метрика. Також його можна вказати тоді, коли створюється метрика. До назв просторів імен є певні вимоги:

Назв можуть містить тільки валідні XML символи.

Розмір назви не може складатися більше ніж з 256 символів

Дозволені символи в назвах – буквено-цифрові символи (0-9A-Za-z), період (.), Дефіс (-), підкреслення (_), коса риска (/), хеш (#) і двокрапка (:).

Те що простори імен у Cloudwatch мають певну угоду назв, що можна спостерігати на прикладі метрик для сервісу AWS EC2 – простір для цього сервісу має назву AWS/EC2.

Метрики є фундаментальною концепцією у Cloudwatch. Вони є упорядкованими за часом набором даних, які потрапляють до системи моніторингу із зовнішніх ресурсів та сервісів. Наприклад, метрика, що стосується використання процесора EC2 інстансом називається CPU Utilization та надається сервісом AWS EC2. Самі набори даних можуть надаватися з різних джерел.

У цілому всі AWS сервіси надсилають свої метрики до Cloudwatch. Метрики від програмного забезпечення, яке не є чистиною AWS, можуть бути надіслані у будь-якому поряду і для них також будуть доступні статистичні дані.

AWS у середині має поділ на регіони, тобто ресурси, які створюються за допомогою цього провайдеру існують у своєму регіоні. Те саме можна сказати і про метрики Cloudwatch – вони існують лише в регіоні, в якому вони були створені. Показники не можуть бути видалені, але вони автоматично видаляються через 15 місяців, якщо не було опубліковано нових даних. Дані старше 15 місяців видаляються на постійній основі. З появою нових даних, дані старше 15 місяців видаляються.

Метрики однозначно визначаються ім'ям, простором імен і виміром. Кожна точка даних у метриці має мітку часу, та (необов'язково) одиницю виміру.

Кожна частина даних метрики повинна бути пов'язана з міткою часу. Мітка часу може становити до двох тижнів у минулому і до двох годин у майбутньому. Якщо не надається мітка часу, Cloudwatch створює мітку часу на основі часу отримання даних.

Мітки часу є об'єктами dateTime, з повними датою, годиною, хвилиною і секундою (наприклад, 2016-10-31T23: 59: 59Z). AWS рекомендує використовувати час у форматі UTC для міток.

Критичні повідомлення перевіряють показники на основі поточного часу в UTC. Користувацькі метрики, що надсилаються до CloudWatch з мітками часу, відмінними від поточного часу UTC, можуть призвести до помилкових спрацювань системи моніторингу або навпаки до спрацювання із затримками.

Cloudwatch має політику зберігання даних, яка впливає на їх гранулярність:

1) дані з періодом менше 60 секунд доступні протягом 3 годин.
2) дані з періодом 60 секунд доступні протягом 15 днів.
3) дані з періодом 300 секунд доступні протягом 63 днів.
4) дані з періодом менше 3600 секунд доступні протягом 3 годин

Метрики, які спочатку опубліковані з більш коротким періодом, об'єднуються для довгострокового зберігання. Наприклад, якщо збирати дані за 1 хвилину, дані залишаються доступними протягом 15 днів з 1-хвилинною гранулярністю. Через 15 днів ці дані все ще доступні, але агрегуються і витягуються тільки з гранулярністю 5 хвилин. Після 63 днів дані додатково агрегуються і доступні з гранулярністю1 година.

Вимір - пара імен / значень, яка є частиною метрики. Для метрики можна призначити до 10 вимірів.

Кожна метрика має специфічні характеристики, які описують її. Виміри допомагають створити структуру для свого статистичного плану. Оскільки вимір є частиною унікального ідентифікатора для метрики, кожен раз, коли додається унікальна пара ім'я / значення до одної з метрик, створюється нова варіація цього показника.

Сервіси AWS, які передають дані до CloudWatch, додають виміри до кожної метрики. Можна використовувати розміри для фільтрації результатів, які повертає CloudWatch. Наприклад, можна отримати статистику для конкретного екземпляра EC2, вказавши вимір InstanceId під час пошуку метрик.

Для метрик, створених певними службами AWS, таких як Amazon EC2, CloudWatch може агрегувати дані за вимірами. Наприклад, якщо шукати метрики в просторі імен AWS / EC2, але не вказуєте жодних вимірів, то CloudWatch агрегує всі дані для вказаної метрики, щоб створити статистику. CloudWatch не об'єднує виміри для користувацьких метрик.

CloudWatch розглядає кожну унікальну комбінацію вимірів як окрему метрику. Статистику можна отримувати лише за допомогою комбінацій вимірів, які спеціально опубліковані. Щоб отримати статистику, вказуються значення для простору імен, назви метрики та назви вимірів, які використовувалися при створенні метрик. Також вказуються час початку та закінчення для агрегації.

Наприклад, припустимо, публікуються різні метрики з іменами ServerStats у просторі імен DataCenterMetric з такими властивостями:

Таблиця 1.1 – Виміри CloudWatch.

Виміри	Юніт	Timestamp	Значення
Server=Prod, Domain=Frankfurt	Count	2016-10-31T12:30:00Z	106
Server=Beta, Domain=Frankfurt	Count	2016-10-31T12:30:00Z	99
Server=Prod, Domain=Rio	Count	2016-10-31T12:30:00Z	189
Server=Beta, Domain=Rio	Count	2016-10-31T12:30:00Z	90

При пошуку даних метрик, їх пошук може бути здійснений с такими комбінаціями вимірів:

1) Server=Prod,Domain=Frankfurt;
2) Server=Prod,Domain=Rio;
3) Server=Beta,Domain=Frankfurt;
4) Server=Beta,Domain=Rio;

Пошук з неповним переліком вимірів не буде успішним. Наприклад, пошук з такими вимірами не поверне даних:

1) Server=Prod;
2) Server=Beta;
3) Domain=Frankfurt;
4) Domain=Rio;

Статистика є агрегацією метрик за певні проміжки часу. CloudWatch надає статистику на основі метрик, наданих користувацьким програмним забезпеченням або наданих іншими службами AWS для CloudWatch. Агрегації здійснюються з використанням простору імен, назви метрики, вимірів і одиниці вимірювання даних протягом вказаного періоду часу. Наступна таблиця описує наявні статистики:

Таблиця 1.2 – Статистики CloudWatch.

Статистика	Опис
Minimum	Найменше значення протягом зазначеного періоду. Це значення можна використовувати для визначення низьких обсягів активності для ресурсів.
Maximum	Найбільш значення протягом зазначеного періоду. Це значення можна використовувати для визначення високих обсягів активності для ресурсів.
Sum	Усі значення, надані для відповідної метрики, додані разом. Ця статистика може бути корисною для визначення загального обсягу використання ресурсів.
Average	Значення Sum / SampleCount протягом вказаного періоду. Порівнюючи цю статистику з мінімумом і максимумом, можна визначити повний обсяг метрики і наскільки близький середній показник до мінімуму і максимуму. Це порівняння допоможе дізнатися, коли потрібно збільшити або зменшити ресурси
SampleCount	Кількість даних, що використовуються для статистичного обчислення.
pNN.NN	Значення процентиля. Ви можете вказати будь-який процентиль, використовуючи до двох знаків після коми (наприклад, p95.45). Статистичні дані про процентні ставки не доступні для показників, які містять будь-які негативні значення.

Кожна статистика має одиницю виміру. Приклади одиниць включають байти, секунди, відсотки тощо.

Під час створення користувацької метрики можна вказати одиницю виміру. Якщо не вказати одиницю виміру, CloudWatch використовує параметр None.

Дані, які визначають одиницю виміру, агрегуються окремо. Коли отримується статистику без вказівки одиниці, CloudWatch об'єднує всі точки даних одного і того ж блоку разом. Якщо є дві однакові метрики з різними одиницями вимірами, повертаються два окремі потоки даних, по одному для кожної.

Період - це період часу, пов'язаний з певною статистикою Amazon CloudWatch. Кожна статистика являє собою сукупність даних метрик, зібраних за певний період часу. Періоди визначаються у секундах, а допустимі значення для періоду становлять 1, 5, 10, 30 або будь-які кратні 60. Наприклад, щоб визначити період шести хвилин, використовується 360 як значення періоду. Можна налаштувати спосіб агрегування даних, змінюючи тривалість періоду. Період може бути коротким як одна секунда або до одного дня (86,400 секунд). Значення за замовчуванням - 60 секунд.

Коли отримується статистика, можна вказати період, час початку і час закінчення. Ці параметри визначають загальну тривалість часу, пов'язану зі статистикою. Значення за замовчуванням для початкового та кінцевого часу надають статистику за останню годину. Значення, які вказуються для початкового та кінцевого часу, визначають, скільки періодів повертається з CloudWatch. Наприклад, отримання статистики з використанням значень за замовченням за період, час початку та час завершення повертає агрегований набір статистичних даних для кожної хвилини попередньої години. Якщо необхідна статистика, агрегованій в десятихвилинних блоках, вказується період у 600. Для статистики, зібраної протягом всієї години, вказується період 3600.

Коли статистичні дані агрегуються протягом певного періоду часу, вони маркуються з часом, відповідним початку періоду. Наприклад, дані, зібрані з 7:00 вечора до 8:00 вечора, мають штамп з 7:00 вечора. Окрім того, дані, зібрані між 7:00 вечора та 8:00 вечора, стають видимими о 7:00 вечора, після чого значення агрегованих даних можуть змінюватися, оскільки CloudWatch збере протягом періоду.

Періоди також важливі для тривог CloudWatch. Коли створюється нагадування, щоб відстежувати певну метрику, CloudWatch передається запит про те, щоб порівняти цей показник із вказаним пороговим значенням. Користувач має великий контроль над тим, як CloudWatch робить це порівняння. Можна не тільки вказати період, протягом якого проводиться порівняння, але також можна вказати, скільки оціночних періодів використовуються для отримання висновку. Наприклад, якщо вказати три періоди оцінки, CloudWatch порівнює вікно з трьома порціями даних. Для показників, які постійно змінюються, CloudWatch не створює тривоги, доки не знайдуться три перевищення порогового значення.

Amazon CloudWatch агрегує статистику відповідно до довжини періоду, який вказується при отриманні статистики. Можна публікувати стільки порцій даних, скільки треба, з тими ж або подібними мітками часу. CloudWatch агрегує їх за довжиною періоду. Агрегована статистика доступна тільки при використанні детального моніторингу. Крім того, Amazon CloudWatch не агрегує дані по регіонах.

Можна опублікувати порції даних для метрики, що використовують не тільки один і той же штамп часу, але й той самий простір і виміри. CloudWatch повертає агреговані статистичні дані для них. Можна також опублікувати декілька порції даних для тих самих або різних метрик, з будь-яким штампом часу.

Для великих наборів даних можна вставити попередньо агрегований набір даних, який називається статистичним набором. За допомогою статистичних наборів для CloudWatch надаються значення Min, Max, Sum і SampleCount для порції даних. Це зазвичай використовується, коли потрібно збирати дані багато разів за хвилину. Припустимо, є показник для затримки запиту на веб-сторінці. Немає сенсу публікувати дані з кожним хітом веб-сторінки. Пропонується зібрати затримку всіх переходів на цю веб-сторінку раз на хвилину та надіслати цю статистику до CloudWatch.

Amazon CloudWatch не розрізняє джерело метрики. Якщо публікується метрика з тим самим простором імен і розмірами з різних джерел, то CloudWatch розглядає це як єдиний показник. Це може бути корисним для метрик послуг у розподіленій, масштабованій системі. Наприклад, всі хости в додатку веб-сервера могли б опублікувати однакові метрики, що представляють затримку запитів, які вони обробляють. CloudWatch розглядає їх як єдиний показник, що дозволяє отримувати статистику для мінімальних, максимальних, середніх і суми всіх запитів до хосту.

Процентиль вказує відносну репутацію значення в наборі даних. Наприклад, 95-й процентиль означає, що 95% даних нижче, ніж це значення, і 5% даних вище, ніж це значення. Процентні знаки допомагають краще зрозуміти розподіл даних метрики. Можна використовувати процентилі з такими сервісами:

1) AWS API Gataway;
2) Elastic Load Balancing;
3) AWS EC2;
4) AWS Kinesis;
5) Amazon RDS;

Часто для виділення аномалій у поведінці програмного забезпечення використовуються процентилі. У типовому розподілі 95% даних знаходяться в межах двох стандартних відхилень від середнього значення, а 99,7% даних знаходяться в межах трьох стандартних відхилень від середнього. Будь-які дані, які виходять за межі трьох стандартних відхилень, часто вважаються аномалією, оскільки вони сильно відрізняється від середнього значення. Наприклад, припустимо, контролюється використання процесора у екземплярах EC2, щоб гарантувати, що клієнти мають хороший user experience. Якщо стежити за середнім показником, це може приховати аномалії. Якщо стежити за максимумом, одна аномалія може призвести до помилкової оцінки результатів. Використовуючи процентилі, можна контролювати 95-й відсотки використання ЦП, щоб перевірити випадки з надзвичайно великим навантаженням.

Статистичні дані про процентилі доступні для користувацьких метрик, а також для метрик сервісів AWS, якщо відправляються необроблені, нерозміщені порції даних. Статистичні дані про процентилі не доступні для метрик, коли будь-яке значення метрики є від'ємним числом.

Для того щоб відслідковувати метрики, які надходять до AWS Cloudwatch можна створювати тривоги, які будуть переходити до критичного стану, коли метрики перевищила порогове значення.

Можна використовувати тривоги для автоматичного ініціювання дій від вашого користувача. Тривога спостерігає за одною метрикою протягом певного періоду часу і виконує одне або кілька заданих дій, виходячи з значення метрики відносно порогового значення протягом часу. Дія - це сповіщення, надіслане темі Amazon SNS або політика автоматичного масштабування. Також можна додавати тривоги до панелей (дашбордів).

Тривоги викликають дії лише для постійних змін стану. Сигнали CloudWatch не викликають дії просто тому, що вони знаходяться в певному стані. Стан повинен змінюватися і підтримуватися протягом певного періоду часу.

Під час створення тривоги обирається період, який є більшим або дорівнює частоті метрики, яку слід контролювати. Наприклад, базовий моніторинг для Amazon EC2 забезпечує відслідковування метрик для екземплярів кожні 5 хвилин. При встановленні тривоги на основній метрики моніторингу обирається період не менше 300 секунд (5 хвилин). Детальний моніторинг для Amazon EC2 забезпечує метрик для екземплярів кожну 1 хвилину. При встановленні сигналу на детальній метриці моніторингу обирається період не менше 60 секунд (1 хвилина).

Якщо встановити тривогу на метрику з високою роздільною здатністю, можна вказати тривогу з періодом 10 секунд або 30 секунд, або можна встановити звичайний тривогу тривалістю в 60 секунд.

Як вказано вище, для сповіщення про появу тривоги у системі, використовується сервіс AWS SNS. Amazon Simple Notification Service (Amazon SNS) - це веб-служба, яка координує та керує доставкою або відправленням повідомлень до кінцевих точок або клієнтів, що підписуються. У Amazon SNS існує два типи клієнтів - видавці та абоненти, які також називаються виробниками та споживачами. Видавці спілкуються асинхронно з абонентами, виробляючи та відправляючи повідомлення до теми, яка є логічною точкою доступу та каналом зв'язку. Абоненти (наприклад, веб-сервери, адреси електронної пошти, черги Amazon SQS, функції AWS Lambda) споживають або отримують повідомлення або сповіщення через один з підтримуваних протоколів

(наприклад, Amazon SQS, HTTP / S, електронну пошту, SMS, Lambda), коли вони підписалися на тему. [13]

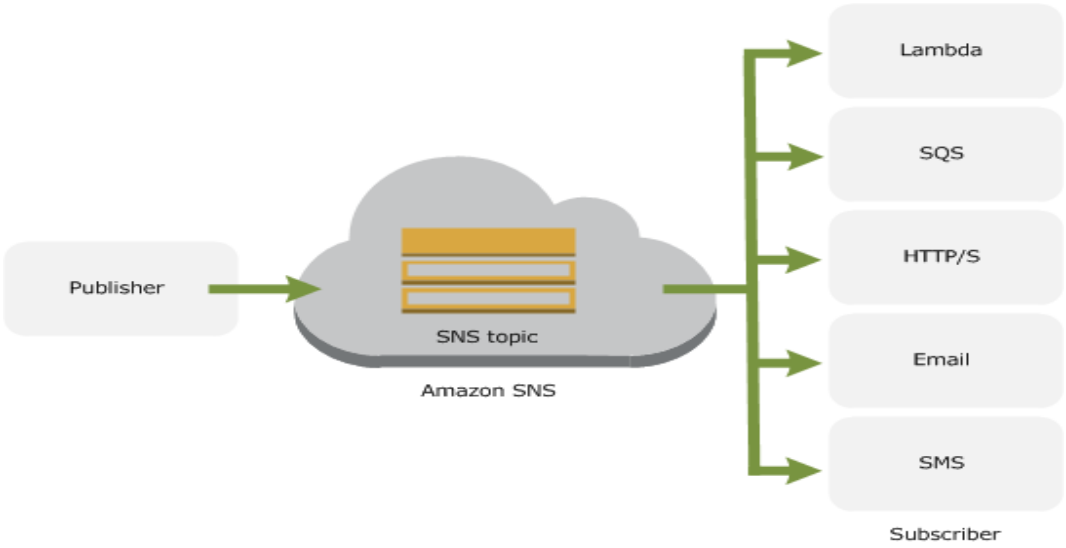

Рис.1.4. – Схеми SNS

Дані, які були отримані та оброблені AWS Cloudwatch, можна переглянути за допомогою Amazon Web Services веб-консолі. Метрики можна переглянути на стандартних дашбордах, тоді треба ввести простір імен, назву метрики та вимір. Також для швидкого доступу до важливих метрик, можна власні дашборди:

Рис.1.5. – Користуватський дашборд Cloudwatch

Також можливо отримати дані з AWS Cloudwatch за допомогою AWS API. Це може бути корисно, коли для відображення метрик використовується якесь користувацьке рішення, наприклад експортують данні з Cloudwatch до іншої системи моніторингу або використовують власне рішення для візуалізації даних та створення тривог.

1.3. Prometheus

1.3.1. Загальний опис

Система моніторингу, створена в компанії SoundCloud та призначена для моніторингу мікросервісів. З моменту свого створення в 2012 році багато компаній і організацій прийняли Prometheus, і проект має дуже активну розробнику і спільноту користувачів. В даний час він є автономним проектом з відкритим кодом і підтримується незалежно від будь-якої компанії. Щоб підкреслити це, а також для уточнення структури управління проектом, Prometheus приєднався до Cloud Native Computing Foundation.

Основними рисами є:

1) багатовимірна модель даних з даними часових рядів, ідентифікованими за назвою метрики і парами ключ / значення
2) PromQL, гнучка мова запитів
3) збір часових рядів відбувається за допомогою моделі pull на HTTP
4) Наявність графічного режиму для відображення метрик.
5) Використання так агентів (експортерів) для збирання метрик.

Prometheus добре працює для запису будь-яких чисто числових часових рядів. Він відповідає як машинно-орієнтованому моніторингу, так і моніторингу високодинамічних сервісно-орієнтованих архітектур. У світі мікросервісів його підтримка для багатовимірного збору даних і запитів є особливо корисною.

Prometheus призначений для надійності. Кожен сервер Prometheus є автономним, не залежно від мережевого сховища або інших віддалених служб. На нього можна покластися, коли інші частини інфраструктури порушені, оскільки не потрібно налаштовувати велику інфраструктуру для використання даної системи моніторингу.

1.3.2. Архітектура Prometheus.

Загальна схема архітектури Prometheus вказана на рис.1.6:

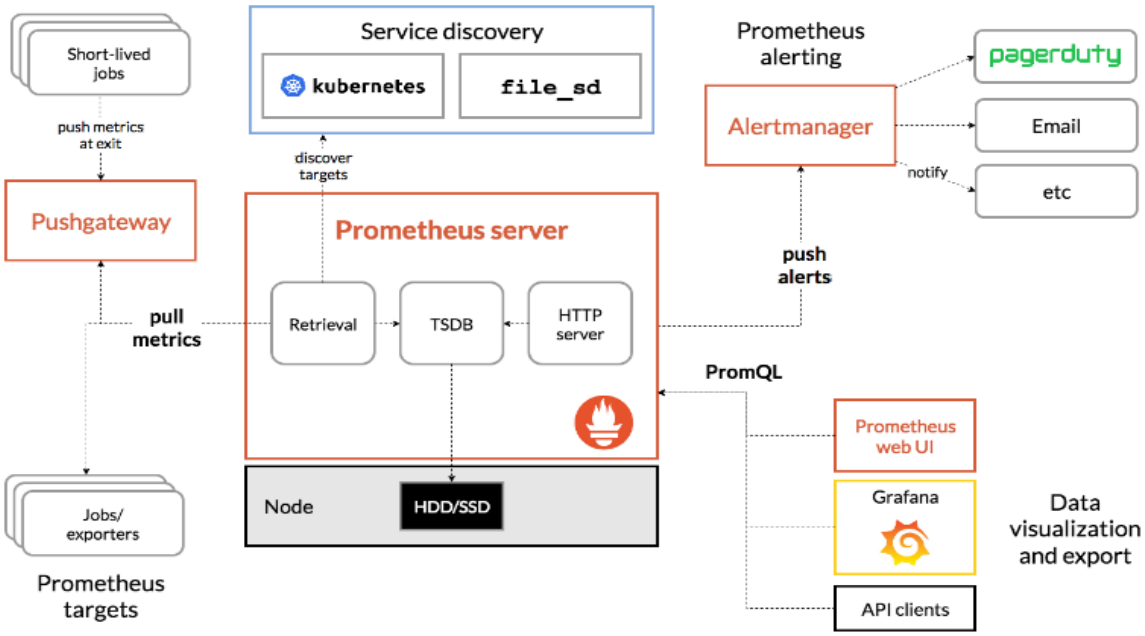

Рис.1.6. – Загальна схема архітектури Prometheus

Основні терміни, що стосуються архітектури Prometheus:

1) Сервер Prometheus - головний сервер, який очищує та зберігає метрики у БД.
2) Scrape - сервер Prometheus використовує метод витягування для отримання метрик.

3) Target - Прометеус обслуговує клієнтів, з яких він отримує інформацію.
4) Експортер - цільові бібліотеки, які конвертують і експортують існуючі метрики у формат Prometheus.
5) Alert manager - компонент, відповідальний за обробку сповіщень.

Сервер Prometheus є центральним компонентом архітектури даної системи моніторингу. Він складається з HTTP-серверу, який віддає веб-консоль систему моніторингу, на якій можна переглянути загальну інформацію про тривоги, поточні тривоги, інформацію про конфігурацію системи моніторингу, список усіх експортерів, які використовуються, а також функціонал по написанню запитів на мові PromQL і побудови графіків для наявних даних. [10]

TSDB – time series data base. База даних, яку використовую Prometheus для зберігання метрик метрик, отриманих від експортерів. Prometheus зберігає всі дані як часові ряди. Окрім збережених часових рядів, Прометей може генерувати тимчасові ряди, отримані тимчасово, як результат запитів.

Кожен часовий ряд однозначно ідентифікується за назвою метрики та набором пар ключ-значення, також відомих як мітки.

Назва метрики вказує загальну характеристику системи, яка вимірюється (наприклад, http_requests_total - загальна кількість отриманих запитів HTTP). Він може містити літери і цифри ASCII, а також символи підкреслення і двокрапку. Він повинен відповідати регулярному виразу [a-zA-Z _:] [a-zA-Z0-9 _:] *.

Двокрапки зарезервовані для правил запису, визначених користувачем. Вони не повинні використовуватися експортерами або прямими приладами.

Мітки дозволяють здійснити розподілену модель даних Прометея: будь-яка комбінація міток для одного і того ж метричного імені визначає

конкретну реалізацію метрики (наприклад: усі HTTP-запити, які використовували метод POST для обробника / api / tracks). Мова запитів дозволяє фільтрувати та агрегувати на основі цих вимірів. Зміна будь-якого значення мітки, включаючи додавання або видалення мітки, створить новий часовий ряд.

Імена міток можуть містити літери ASCII, цифри, а також підкреслення. Вони повинні відповідати регулярному виразу [a-zA-Z _] [a-zA-Z0-9 _] *. Назви міток, які починаються з __, зарезервовані для внутрішнього використання.

Значення мітки можуть містити будь-які символи Unicode.

При відомих значеннях іменні метрики та її міток, загальна часовий ряд часто ідентифікується за допомогою такого виразу:

<metric name>{<label name>=<label value>, ...}

Наприклад для метрики api_http_requests_total, для якої встановлені мітки method="POST" та handler="/messages" часовий ряд бути мати такий вигляд:

api_http_requests_total{method="POST", handler="/messages"}

Retrieval – це блок, який відповідає за опит зовнішніх компонентів, таких як експортери, які постачають порції даних до системи моніторингу. Оскільки, комунікація відбувається за допомогою протоколу http, єдиною вимогою до реалізації цих експортерів є відповідність формату даних, яку вони віддають системі моніторингу.

Prometheus може також забирати данні не тільки від своїх експортерів, а також і від програмного забезпечення напряму.

Клієнтські бібліотеки Prometheus пропонують чотири основні типи метрик. Наразі вони розрізняються лише в клієнтських бібліотеках (для включення API, пристосованих до використання конкретних типів). Сервер Prometheus ще не використовує інформацію про тип і вирівнює всі дані в нетипізовані часові ряди. Це може змінитися в майбутньому.

Клієнтькі бібліотеки для Prometheus доступні для таких мов програмування:

- Go
- Java
- Ruby
- Python

Лічильник - це кумулятивний показник, який представляє собою єдиний монотонно зростаючий лічильник, значення якого може тільки збільшуватися або повертатися до нуля при перезапуску. Наприклад, можна використовувати лічильник для представлення кількості поданих запитів, завершених завдань або помилок.

Не треба використовувати лічильник для розкриття значення, яке може зменшитися. Наприклад, не треба використовувати лічильник для кількості поточних процесів; замість цього краще використати датчик.

Датчик - це показник, який представляє собою одне числове значення, яке може довільно йти вгору і вниз.

Датчики зазвичай використовуються для вимірюваних значень, таких як температура або поточне використання пам'яті, а також "підрахунок", який може йти вгору і вниз, як кількість одночасних запитів.

Гістограма вибирає спостереження (зазвичай такі речі, як тривалість запиту або розмір відповіді) і підраховує їх у конфігураційних ковшах. Він також дає суму всіх спостережуваних значень.

У термінах Prometheus кінцеву точку, яка може надавати даны, називають екземпляром, зазвичай відповідним одному процесу. Колекція екземплярів з тією ж метою, процес, що реплікується для масштабованості або надійності, називається завданням (job).

Конфігурація тривог. Правила тривог дозволяють визначити умови сповіщення на основі виразів мови Prometheus та надіслати сповіщення

про попередження до зовнішньої служби. Всякий раз, коли вираз тривоги призводить до одного або декількох елементів в певний момент часу, попередження вважається активним. Загалом у тривог є в контексті Prometheus є наступні статуси:

- OK – відслідковуєма метрики не перевищила порогового значення.
- Pending – відслідковуєма метрики перевищила порогове значення, проте не перевищила значення періоду.
- Firing – відслідковуєма метрики перевищила порогове значення, та перевищила значення періоду.

Зазвичай при створенні тривог для метрик, проблемою їх тестування, оскільки інженеру треба якимось чином симулювати критичну ситуацію та перевірити відгук системи моніторингу на неї. Це буває досить складно, тому зачасту тривоги системи моніторингу не тестуються взагалі, що може погону вплинути на інфраструктуру. Для вирішення цієї проблеми розробники Prometheus пропонують юніт тести для правил критичних повідомлень. Для здійснення тестуванню цих правил необхідно написати тестові файли з необхідними конфігураціями та використати утиліту promtool, яка доступна у офіціальному репозиторії системи моніторингу на githhub.

Для надсилання повідомлень про тривоги, Prometheus використовує зовнішній сервіс, що має назву Alert Manager. Alert Manager – це веб-сервіс, який виконує функції по збиранню, агрегації критичних повідомлень, а також сповіщення про них різними каналами зв’язку. Alert Manager тільки примає дані про критичні повідомлень, тому для підтримки статусу критичного повідомлення необхідно постійно відправляти у нього відповідні повідомлення, в іншому випадку Alert Manager вважає тривогу виправленню та повідомляє про це. Також Alert Manager підтримує конфігурацію для повідомлень, тобто можливо

виставляти отримувачів, часи отримання, конфігурувати правила ігнорування певних критичних повідомлень. Alert manager має свої API, який може бути використаний, наприклад, для тестування інтеграцій з іншими каналами зв'язку.

Хоча Prometheus використовує Pull-модуль, є також спосіб надіслати у нього метрики за допомогою Prometheus Pushgateway.

Prometheus також має також свій Rest API. Більшість доступних ендпоінтів дозволяє переглянути інформації про поточну конфігурації системи моніторингу – наприклад, можливо переглянути поточні тривоги, загальну інформацію про метрики, для яких заданні порогові значення, отримати список таргетів тощо. Також можливо виконати запит до щодо метрики за допомогою API. Це використовується у першу чергу для візуалізації даних іншими сервісами, такими як Grafana.

Хоча більшість ендпоінтів у Prometheus API є read-only, тобто дозволяють переглянути поточні метрики або конфігурації, є також так звана TSDB Admin API, яка стає доступна після використання опції «--web.enable-admin-api» і дозволяє:

- Зробити снапшот поточних даних та записати на диск. У відповідь приходить назва директорії, де був збережений снапшот та його назва.
- Видалити певний часовий ряд з бази.
- Видалити дані, які все не можуть бути використані системою моніторингу.

1.4. Zabbix.

1.4.1. Загальний опис

Zabbix - це мережеве програмне забезпечення для корпоративного моніторингу для будь-якої IT-інфраструктури, додатків, сервісів і ресурсів. Це рішення з відкритим кодом, створене для моніторингу в

реальному часі мільйонів показників, зібраних з декількох мережевих пристроїв, серверів і віртуальних машин. Платформа дозволяє користувачам збирати та аналізувати показники та статистику продуктивності, візуалізувати її та отримувати сповіщення про поточні та майбутні проблеми оперативно.

Рішення надає надійні інструменти та функції, які контролюють все в мережі від процесорів і серверів до IT-додатків, мережевих пристроїв і баз даних. Одним з найпопулярніших модулів є агент Zabbix, який є динамо-рішенням з можливостями управління інфраструктурою. Модуль пропонує надійні засоби моніторингу мережі, які дозволяють користувачам розробити стратегію розширення потужності з поглибленими показниками використання мережі.

Zabbix складається з трьох базових компонентів:

1. сервера для координації виконання перевірок, формування перевірочних запитів та накопичення статистики;
2. агентів для здійснення перевірок на стороні зовнішніх хостів;
3. фронтенда для організації управління системою.

Для зняття навантаження з центрального сервера і формування розподіленої мережі моніторингу може бути розгорнута серія проксі-серверів, котрі агрегують дані про перевірку групи хостів. Сирцевий код агентів і серверної частини написаний на мові Сі, для розробки веб-інтерфейсу використано мову PHP, дані можуть зберігатися в СУБД MySQL, PostgreSQL, SQLite, DB2 і Oracle. Код проекту поширюється під ліцензією GPLv2. [9]

Загалом достатньо популярна система моніторингу, має такі переваги: сценарії на основі моніторингу, підтримка SNMP, гнучка система шаблонів і груп, підтримка моніторингу JMX, звітність і тенденції.

Панель керування Zabbix зображено на рис.1.7.:

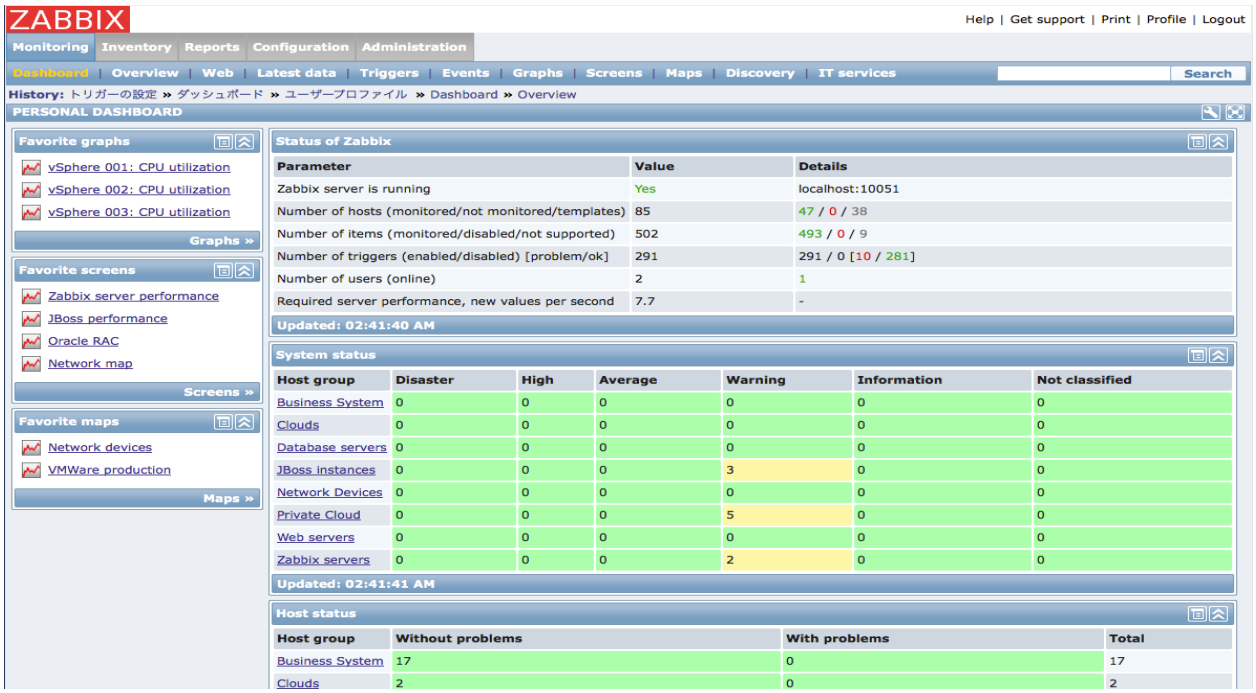

Рис.1.7. – Панель керування Zabbix

1.4.2. Архітектура Zabbix.

Загальна схема архітектури Zabbix вказана на Рис.1.8:

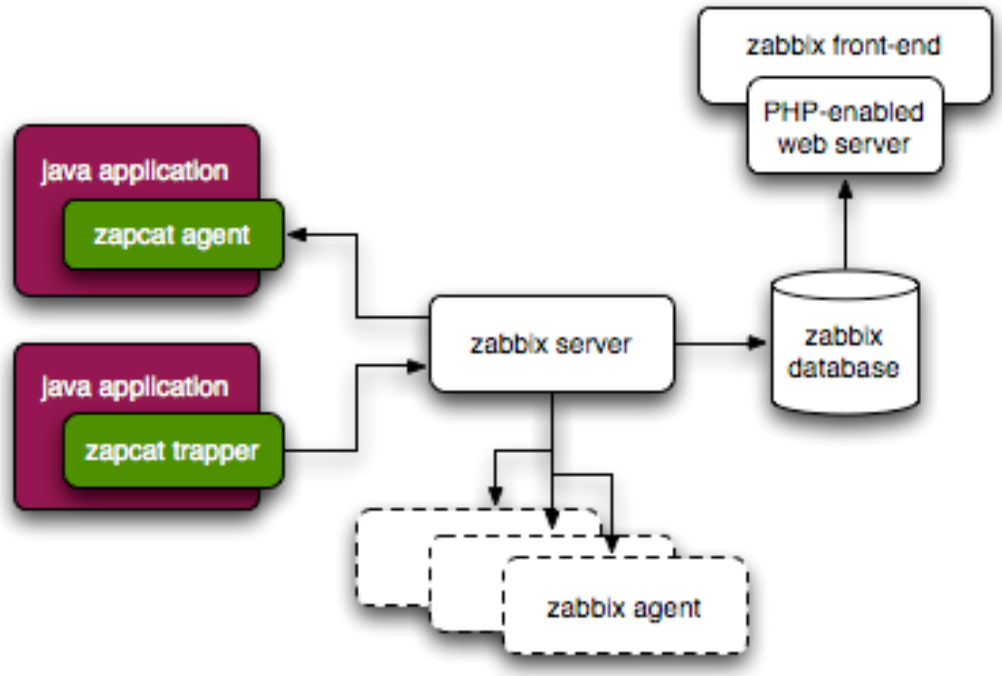

Рис.1.8. - Загальна схема архітектури Zabbix

Сервер Zabbix є центральним процесом програмного забезпечення Zabbix.

Сервер виконує опитування та захоплення даних, обчислює тригери, надсилає повідомлення користувачам. Це центральний компонент, до якого агенти Zabbix і проксі повідомляють дані про доступність і цілісність систем. Сервер може самостійно перевіряти мережеві служби (такі як веб-сервери та поштові сервери), використовуючи прості сервісні перевірки.

Сервер є центральним сховищем, в якому зберігаються всі конфігураційні, статистичні та операційні дані, і це об'єкт в Zabbix, який буде активно оповіщати адміністраторів, коли виникають проблеми в будь-якій з контрольованих систем.

Функціонування основного сервера Zabbix розбивається на три окремі компоненти; це: сервер Zabbix, веб-інтерфейс і сховище баз даних.

Вся інформація про конфігурацію Zabbix зберігається в базі даних, з якою взаємодіють і сервер, і веб-інтерфейс. Наприклад, коли створюється новий елемент за допомогою веб-інтерфейсу (або API), його додають до таблиці елементів у базі даних. Потім приблизно раз на хвилину сервер Zabbix запитую таблицю елементів для списку активних елементів, які потім зберігаються в кеші на сервері Zabbix. Ось чому може знадобитися до двох хвилин для того, щоб будь-які зміни, внесені в інтерфейс Zabbix, відображалися в останньому розділі даних.

Сервер Zabbix розроблений для роботи як не root користувача. Він запускатимється з будь-яким не root користувачем. Таким чином, можна запускати сервер як будь-який користувач, що не є root, без будь-яких проблем.

Якщо спробувати запустити його як "root", він переключиться на користувача "zabbix", який повинен бути присутнім у системі.

Ці налаштування в даний час не можуть бути налаштовані користувачем, ні під час компіляції, ні в конфігураційному файлі.

Якщо сервер Zabbix і агент виконуються на одній машині, рекомендується використовувати іншого користувача для запуску сервера, і для запуску агента. В іншому випадку, якщо обидва виконуються від імені одного і того же самого користувача, агент може отримати доступ до файлу конфігурації сервера, і будь-який користувач рівня адміністратора в Zabbix може досить легко отримати, наприклад, пароль бази даних.

Через вимоги безпеки та критично важливого характера роботи сервера, UNIX є єдиною операційною системою, яка може послідовно забезпечувати необхідну продуктивність і відмовостійкість для роботи серверу в умовах високонавантаженої інфраструктури. Крім того, документація написана з прикладами використання для UNIX.

Сервер Zabbix може працювати на таких платформах:

1) Linux;
2) Solaris;
3) AIX;
4) HP-UX;
5) Mac OS X;
6) FreeBSD;
7) OpenBSD;
8) NetBSD;
9) SCO Open Server;
10) Tru64/OSF1;

Агент Zabbix розгортається для активного спостереження за локальними ресурсами та додатками (жорсткі диски, пам'ять, статистика процесорів тощо).

Агент збирає оперативну інформацію локально і передає дані на сервер Zabbix для подальшої обробки. У разі збоїв, сервер Zabbix може активно сповіщати адміністраторів конкретної машини, яка повідомила про помилку.

Агенти Zabbix надзвичайно ефективні через використання системних викликів для збору статистичної інформації.

Агенти Zabbix можуть виконувати пасивні та активні перевірки.

При пасивній перевірці агент реагує на запит від центрального вузла. Сервер Zabbix (або проксі) запитує дані, наприклад, завантаження процесора, а агент Zabbix відправляє результат.

Активні перевірки вимагають більш складної обробки. Агент повинен спочатку отримати список елементів із сервера Zabbix для незалежної обробки. Потім він періодично надсилатиме нові значення на сервер. Такий спосіб є більш еластичним, оскільки дозволяє проводити моніторинг віддалених вузів, які знаходяться в інших АВ та не мають власного публічного IP та недоступні «зовні». Проте, це вимагає більш складної побудови інфраструктури. На приклад, коли Zabbix сервер розгорнутий за допомогою docker, з використанням bridge мережевого режиму з рандомним портом, необхідно застосувати механізм service discovery, для того щоб агент міг успішно контактувати з Zabbix сервером.

Виконання пасивних або активних перевірок налаштовується шляхом вибору відповідного типу елемента моніторингу. Агент Zabbix обробляє елементи типу "агент Zabbix" або "агент Zabbix (активний)"

На відміну від Zabbix сервера, Zabbix агент можна розгорнути під ОС Windows.

Агент Zabbix на Windows працює як служба Windows. Можна запустити один екземпляр агента Zabbix або кілька екземплярів агента на хості. Один примірник може використовувати файл конфігурації за

замовчуванням або файл конфігурації, вказаний у командному рядку. У випадку декількох екземплярів кожен екземпляр агента повинен мати свій власний файл конфігурації (один з екземплярів може використовувати файл конфігурації за замовчуванням).

Агент вимагає локалі UTF-8, щоб деякі текстові елементи агента могли повернути очікуваний вміст. У більшості сучасних Unix-подібних систем за замовчуванням існує локалізація UTF-8, однак існують деякі системи, в яких може знадобитися додаткова конфігурація.

Можна дозволити активну автореєстрацію агента Zabbix, після чого сервер може почати їх моніторинг. Таким чином, нові хости можуть бути додані для моніторингу без їх ручного налаштування на сервері.

Автоматична реєстрація може статися, коли раніше невідомий активний агент запитує перевірки.

Ця функція може бути дуже зручною для автоматичного моніторингу нових вузлів. Як тільки з'явиться новий вузол, Zabbix автоматично запустить збір даних про продуктивність і доступність хоста.

Активна автореєстрація агентів також підтримує моніторинг доданих хостів за допомогою пасивних перевірок. Коли активний агент запитує перевірки, якщо у конфігураційному файлі визначено параметри конфігурації "ListenIP" або "ListenPort", вони надсилаються на сервер. (Якщо вказано кілька IP-адрес, перший надсилається на сервер.)

Сервер, додаючи новий авто-зареєстрований хост, використовує отриману IP-адресу та порт для налаштування агента. Якщо значення IP-адреси не отримано, використовується значення, яке використовується для вхідного з'єднання. Якщо значення порту не отримано, використовується 10050.

Zabbix proxy - це процес, який може збирати дані моніторингу з одного або декількох контрольованих пристроїв і надсилати інформацію

на сервер Zabbix, по суті працюючи від імені сервера. Всі зібрані дані буферизуються локально, а потім передаються на сервер Zabbix, до якого належить проксі-сервер.

Розгортання проксі-сервера є необов'язковим, але може бути дуже корисним для розподілу навантаження одного сервера Zabbix. Якщо тільки проксі збирають дані, обробка на сервері стає меншою.

Проксі Zabbix є ідеальним рішенням для централізованого моніторингу віддалених локацій, філій і мереж без місцевих адміністраторів.

Проксі-сервер Zabbix вимагає окремої бази даних.

Zabbix proxy може бути розгорнутий на тих самих платформах, що і Zabbix сервер.

Відправник Zabbix - це утиліта командного рядка, яка може використовуватися для передачі даних про продуктивність на сервер Zabbix для обробки.

Зазвичай, утиліта використовується для тривалих користувацьких скриптів для періодичної передачі даних про доступність і продуктивність.

1.5. Висновки до розділу 1

У першому розділі бакалаврської роботи виконано огляд систем моніторинг, з якими у подальшому виконується інтергація для збирання критичних повідомлень у системі. Було розглянуто основні принципи їх архітектури, основні інструменти та залежності, які використовуються ними для проведення моніторингу, розглянуто переваги та недоліки кожної з них. У результаті можна зробити висновки:

1. Розглянуто основні терміни, архітектурні особливості системи моніторингу Cloudwatch
2. Розглянуто основні терміни, архітектурні особливості системи моніторингу Zabbix
3. Розглянуто основні терміни, архітектурні особливості системи моніторингу Prometheus

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО КОМПЛЕКСУ ДЛЯ ОТРИМАННЯ КРИТИЧНИХ ПОВІДОМЛЕНЬ ВІД РІЗНИХ СИСТЕМ МОНІТОРИНГУ

2.1. Опис використаних технологій

Для виконання системи агрегації критичних повідомлень від різних систем моніторингу була обрана мова програмування Python. Перелік використаних модулів:

- Python Flask, версія 1.0.2
- Python boto3, версія 1.9.112
- Python requests, версія 2.22.0

2.1.1. Загальні відомості про Python.

Python - це високорівнева, інтерпретована, інтерактивна та об'єктно-орієнтована мова сценаріїв. Python розроблений так, щоб його можна було легко читати. Він часто використовує ключові слова англійської мови, де інші мови використовують пунктуацію, і він має менше синтаксичних конструкцій, ніж інші мови.

Основні особливості:

1) Python обробляється під час виконання інтерпретатором. Перед виконанням програми не потрібно компілювати програму. Це схоже на PERL і PHP.
2) Python забезпечує підвищену читабельність. Для цього єдині відступи використовуються для розмежування блоків операторів замість фігурних дужок, як на багатьох мовах, таких як C, C ++ і Java.
3) Підтримує такі об'єктно-орієнтовані концепції програмування, як клас, успадкування, об'єкти, модуль, простір імен тощо.

4) Python - це крос-платформна мова. Він працює однаково на різних платформах ОС, таких як Windows, Linux, Mac OSX тощо.
5) Python є безкоштовним і розповсюджується як відкрите програмне забезпечення. Велике програмне співтовариство бере активну участь у розробці та підтримці бібліотек Python для різних додатків, таких як веб-фреймворки, математичні обчислення та наука про дані.

2.1.2. Flask

Flask - це мікро веб-платформа, написана на Python. Він класифікується як мікрфреймворк, оскільки не вимагає спеціальних інструментів або бібліотек. Він не має рівня абстракції бази даних, перевірки форм або будь-яких інших компонентів, на відміну від інших веб-фрейморків. Однак, Flask підтримує розширення, які можуть додавати функції так , як якщо б вони були реалізовані в самому Flask. Існують розширення для об'єктно-реляційних мапперів, валідації форм, обробки завантажень, різних відкритих технологій аутентифікації. Розширення оновлюються набагато більш регулярно, ніж основний веб-фреймворк Flask.

Основні компоненти веб-фреймворку:

- WSGI
- Werkzeug
- Jinga2

WSGI - це специфікація, викладена в PEP 333, для стандартизованого інтерфейсу між веб-серверами і веб-фреймворками / додатками Python.

Мета полягає в тому, щоб забезпечити відносно простий, але всеосяжний інтерфейс, здатний підтримувати всі (або більшість) взаємодій між веб-сервером і веб-середовищем

Додатковою метою є підтримка компонентів «проміжного програмного забезпечення».

Werkzeug – це інструментарій WSGI, який реалізує запити, об'єкти реагування та інші функції утиліти. Це дозволяє створювати веб-фреймворк над нею. Flask використовує Werkzeug як одну з своїх основ.

Jinja2 - це мова сучасних шаблонів для розробників Python. Вона використовується для створення HTML, XML або інших форматів розмітки, які повертаються користувачеві через HTTP-запит.

Jinja2 має такі особливості:

- Режим виконання Sandbox. Це дозволяє виконувати тестові шаблони.
- запобігання крос-сценаріїв.
- Успадкування шаблонів дозволяє використовувати однаковий або схожий базис для всіх шаблонів.

2.1.3. Boto3.

Boto - це SDK Amazon Web Services (AWS) для Python. Вона дозволяє розробникам Python створювати, конфігурувати і керувати службами AWS, такими як EC2 і S3. Boto надає простий у використанні об'єктно-орієнтований API, а також низький рівень доступу до послуг AWS. [8]

Основні компоненти:

- облікові дані та необлікові дані
- ресурси
- низькорівневі клієнти
- пагінатори

Boto можна налаштувати кількома способами. Незалежно від обраного джерела або джерел, для виконання запитів необхідно мати облікові дані AWS та набір регіонів.

У boto3 існують два типи конфігурації: облікові дані та необлікові дані. Облікові дані включають такі елементи, як aws_access_key_id, aws_secret_access_key та aws_session_token. Конфігурація не-облікових даних включає в себе такі елементи, як регіон, який потрібно використовувати, або який стиль адресації використовувати для Amazon S3. Різниця між обліковими даними та конфігурацією необлікових даних є важливою, оскільки процес пошуку дещо відрізняється. Boto3 буде шукати декілька додаткових місць при пошуку облікових даних, які не застосовуються при пошуку конфігурації, що не містить облікових даних.

Ресурси представляють об'єктно-орієнтований інтерфейс для веб-сервісів Amazon (AWS). Вони забезпечують більш високий рівень абстракції, ніж вихідні, низькі виклики, зроблені клієнтами служби. Щоб використовувати ресурси, викликається метод resource () та передається ім'я сервісу. [15]

Кожен екземпляр ресурсу має ряд атрибутів і методів. Їх можна концептуально розділити на ідентифікатори, атрибути, дії, посилання, підресурси та колекції.

Самі ресурси також можна концептуально розділити на сервісні ресурси (наприклад, sqs, s3, ec2 і т.д.) і окремі ресурси (наприклад, sqs.Queue або s3.Bucket). Сервісні ресурси не мають ідентифікаторів або атрибутів.

Клієнти забезпечують інтерфейс низького рівня для AWS, методи якого близькі до API служб. Всі сервісні операції підтримуються клієнтами.

Деякі операції AWS повертають результати, які є неповними і вимагають наступних запитів для отримання всього набору результатів. Процес надсилання наступних запитів продовжується, коли попередній запит припинено, називається pagination. Наприклад, операція list_objects Amazon S3 повертає до 1000 об'єктів одночасно, і необхідно надіслати

наступні запити відповідним маркером, щоб отримати наступну сторінку результатів.

Пагінатори- це особливість boto3, що діє як абстракція над процесом ітерації по всьому набору результатів виклику API.

2.1.4. Requests.

Бібліотека requests є де-факто стандартом для створення HTTP-запитів у Python. Вона абстрагує складність створення запитів за гарним, простим API, так що можна зосередитися на взаємодії з послугами і отримувати дані у додатку.

2.2. Опис веб-серверу

2.2.1. Опис ендпоінтів.

Розроблений веб-сервер має наступні ендпоінти:

- /, method=GET - використовується у якості хелс-чеку програми. Повертає 200 коли веб-сервер стартував успішно
- /query, method=POST – використовується для того, щоб отримати дані про критичні повідомлення від різних систем моніторингу, з якими наявна інтеграція.
- /search, method=POST – використовується для того, щоб отримати список усіх доступних метрик. На даний момент доступні метрики – cloudwatch_green, cloudwatch_red, cloudwatch_insufficient_data, prometheus_red, prometheus_all.

2.2.2.Опис модулів.

На даний момент було розроблено 3 модулі для інтеграції с різними системами моніторингу.

Модуль Cloudwatch. Розроблений для інтеграції з системою моніторингу Cloudwatch. Для інтеграції використовується модулі bot3. Реалізовані методи:

- getAlarm(alarm_state) – повертає список тривог, наявних у системі моніторингу Cloudwach. Alarm_state має три дозвлені значення – ALARM (зробити пошук тривог, які знаходяться у активному стані), INSUFFICIENT_DATA (зробити пошук тривог, дані метрик для яких не отримуються), OK (зробити пошук тривог, які не знаходяться в активному стані).
- table_cloudwatch - повертає результат пошуку у вигляді таблиці.

Модуль Prometheus. Розроблений для інтеграції з системою моніторингу Prometheus. Для інтеграції використовується модулі requests. Реалізовані методи:

- getRules() – повертає список тривог, які сконфігуровані у системі моніторингу Prometheus.
- getAlerts() – повертає список тривог, що знаходяться в активному стані.
- table_prometheus - повертає результат пошуку у вигляді таблиці.

2.3. Результат виконання.

Для візуалізації отриманих даних від різних систем моніторингу був обраний сервіс Grafana.

Grafana - це інструмент візуалізації з відкритим вихідним кодом, який можна використовувати поверх різноманітних сховищ даних, але найчастіше використовується разом з Graphite, InfluxDB, а також Elasticsearch і Logz.io.

По суті, це багатофункціональна заміна Graphite-web, яка допомагає користувачам легко створювати та редагувати інформаційні панелі. Він містить унікальний синтаксичний аналізатор графіту, що дозволяє легко

редагувати метрику та функцію. Користувачі можуть створювати вичерпні діаграми з інтелектуальними форматами осей (наприклад, лініями та точками).

Grafana підтримує різні джерела даних. Для даної роботи був обраний SimpleJsonDataSource, оскільки веб сервер віддає відповіді на запити у json форматі.

Список критичних повідомлень отриманих від систем моніторингу Cloudwatch (Рис.2.1):

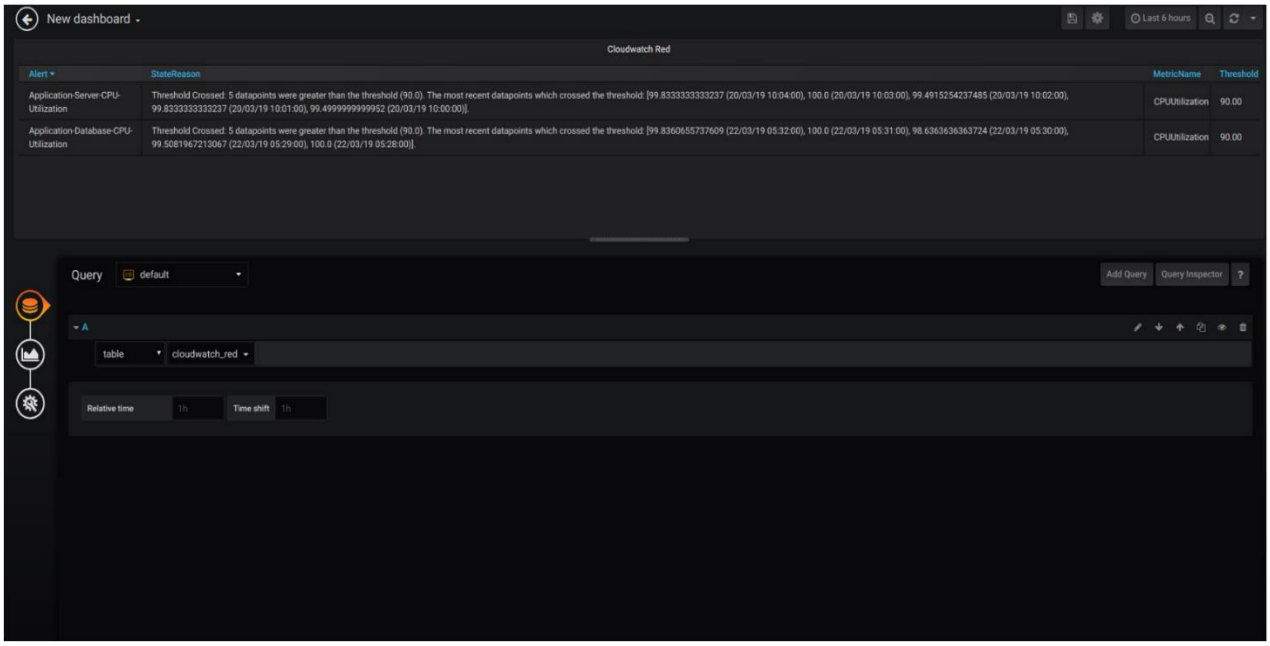

Рис.2.1. Інтерфейс Grafana. Критичні повідомлення Cloudwatch.

Список критичних повідомлень отриманих від системи моніторингу Prometheus (Рис.2.2):

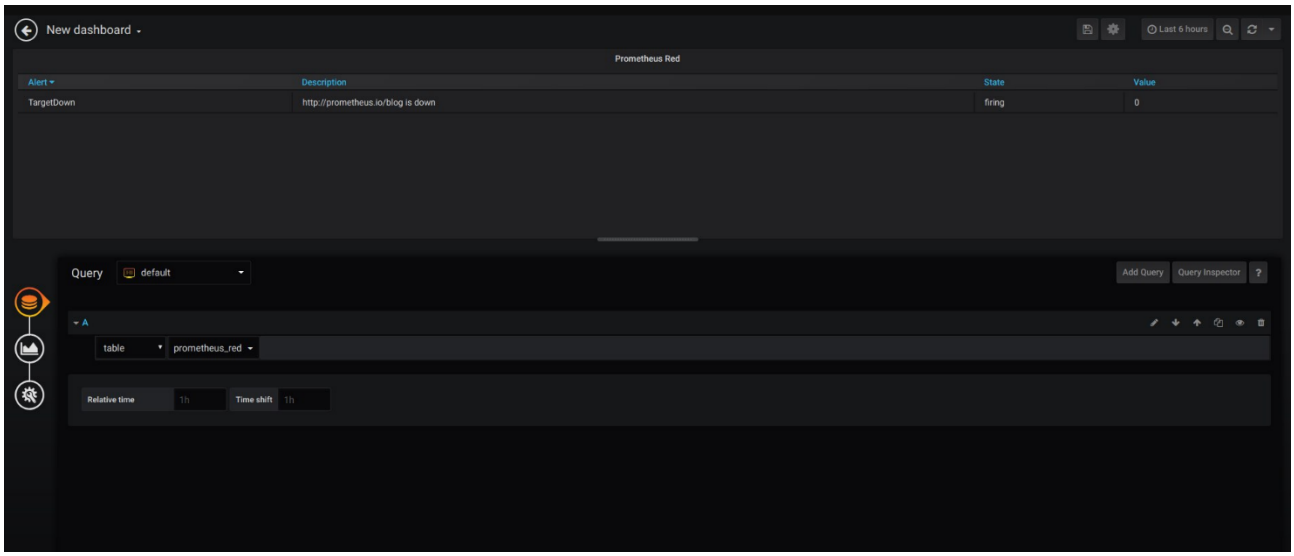

Рис.2.2. Інтерфейс Grafana. Критичні повідомлення Prometheus.

2.4. Висновки до розділу 2

У другому розділі бакалаврської роботи виконано розроблену веб-сервер, який реалізує основний функціонал по інтеграції та збору інформації про критичні повідомлення від різних систем моніторингу . У результаті можна зробити висновки:

4. Розглянуто використанні технології, які використовувалися для розробки веб серверу, наданий детальний опис використаної мови програмування, модулів тощо.
5. Зроблений детальний опис функціоналу розробленого веб-серверу, такого як: список усіх ендпоінтів, очікувані відповіді, які надає система, список дозволених HTTP методів для кожного ендпоінту, зроблений опис розроблених модулів, классів та методів, які вони містять.
6. Приведені результати роботи веб серверу за допомогою інструменту візуалізації Grafana.

РОЗДІЛ 3. РОЗГОРТАННЯ СИСТЕМИ ВІЗУАЛІЗАЦІЇ КРИТИЧНИХ ПОВІДОМЛЕНЬ РІЗНИХ СИСТЕМ МОНІТОРИНГУ

3.1.Середовище розгортання

Для розгортання системи візуалізації критичних повідомлень були реалізовано два варіанти:

1) розгортання на базі платформи AWS;
2) розгортання для локального тестування і розробки з використанням технологій docker та docker-compose.

3.2.Основні вимоги до інфраструктури

3.2.1.HA.

Висока доступність (HA) - це здатність системи або компонента системи постійно працювати протягом довгого часу. Доступність може бути виміряна щодо "100% експлуатації" або "ніколи не виходить з ладу". У сфері інформаційних технологій (IT) широко поширений, але важкодоступний стандарт доступності для системи або продукту відомий як "п'ять 9" (99,999%) доступності. [2]

Експерти підкреслюють, що для того, щоб будь-яка система була високодоступною, частини системи повинні бути добре спроектовані і ретельно перевірені перед їх використанням. Оскільки комп'ютерна система або мережа складається з багатьох частин, в яких всі частини, як правило, повинні бути присутніми для того, щоб все було в робочому стані, велике планування для центрів високої доступності навколо резервного копіювання та обробки після збоїв і зберігання даних і доступу.

Як досягається висока доступність [2]:

1) використання розподілення навантаження та використання декількох примірників сервісів.

2) розгортання сервісів у різних регіонах, щоб недієздатність сервісу в одному регіоні несуттєво вплинула на роботу сервісу в цілому.
3) налаштування надійної системи моніторингу.

3.2.2.Fault Tolerance.

Відмовостійкість - це властивість, яка дозволяє системі продовжувати працювати належним чином у разі виходу з ладу (або одного або декількох несправностей всередині) деяких його компонентів. У високодоступних або життєво важливих системах особливо відмовостійкість. [5]

Відмовостійка інфраструктура дозволяє сервісу продовжувати свою роботу , можливо, на зниженому рівні, якщо якась частина сервісу виходить з ладу. Цей термін найчастіше використовується для опису комп'ютерних систем, призначених для продовження більш-менш повноцінного функціонування з, можливо, зменшення пропускної здатності або збільшення часу відгуку в разі часткової невдачі. Тобто, система в цілому не зупиняється через проблеми або в апаратному, або в програмному забезпеченні. Прикладом в іншій галузі є автомобільний транспортний засіб, сконструйований таким чином, що він буде продовжувати рух, якщо одна з шин проколота, або конструкція, яка здатна зберігати свою цілісність при наявності пошкодження внаслідок таких причин, як корозія, недоліки конструкції, помилки при проектуванні. [5]

В рамках окремої системи допуск відмови може бути досягнутий завдяки передбаченню виняткових умов і побудові системи для їх подолання, і, загалом, прагнення до самостабілізації. Проте, якщо наслідки відмови системи є катастрофічними, або витрати на її достатню надійність дуже високі, кращим рішенням може бути використання певної форми дублювання. У будь-якому випадку, якщо наслідки відмови

системи настільки катастрофічні, система повинна мати можливість використовувати повернення до безпечного режиму.

Відмовостійкість може відігравати певну роль у стратегії відновлення після аварії. Наприклад, відмовостійкі системи з резервними компонентами у хмарній платформі можуть швидко відновлювати критично важливі системи, навіть якщо природні або людські катастрофи знищують IT-інфраструктуру на місці.

3.2.3.Redundancy

Надлишковість- це в основному додаткове обладнання або програмне забезпечення, яке можна використовувати як резервну копію, якщо основне апаратне або програмне забезпечення виходить з ладу. Надмірність може бути досягнута через кластеризацію навантаження, RAID, балансування навантаження, високу доступність в автоматизованому режимі. Більш високий рівень надлишковісті досягається, коли пристрій резервного копіювання повністю відокремлюється від основного пристрою. Наприклад, резервна інтернет-лінія надається іншим провайдером ISP, тому повністю окрема фізична зв'язок і підключення від первинного підключення до Інтернету, або надлишкове обладнання, яке знаходиться в іншій будівлі. [2]

3.2.4.Containerization.

Контейнеризація - це легка альтернатива віртуалізації, яка включає в себе інкапсуляцію програми в контейнері з власною операційною системою. [3]

Найпопулярнішим рішенням для контейнеризації є docker.

Docker - це платформа розробки програмного забезпечення з відкритим вихідним кодом. Його головна перевага полягає в тому, щоб запаковувати програми в контейнери, дозволяючи їм бути портативними

для будь-якого сервісу, що працює під операційною системою Linux або Windows. Машина Windows може запускати контейнери Linux за допомогою віртуальної машини (VM). Загальна схема контейнерів зображена на Рис.3.1. [4]

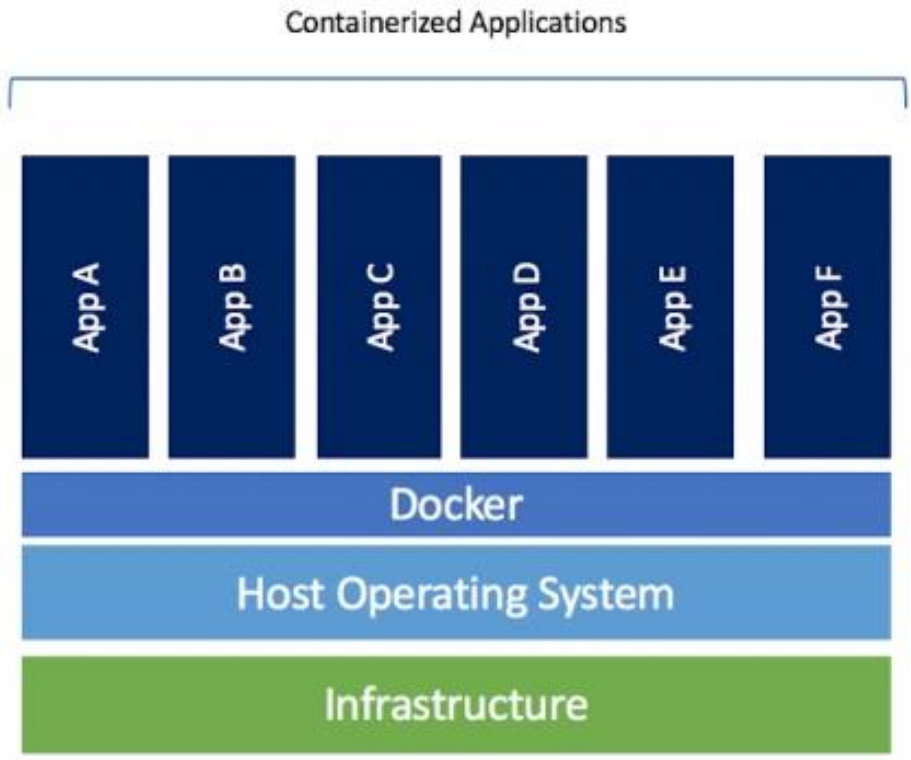

Рис.3.1. - Загальна схема контейнерів

3.3.Розгортання інфраструктури на базі хмарного провайдера.

У якості хмарного провайдера була обраний AWS. Спрошена інфраструктура системи зображена на Рис.3.2.:

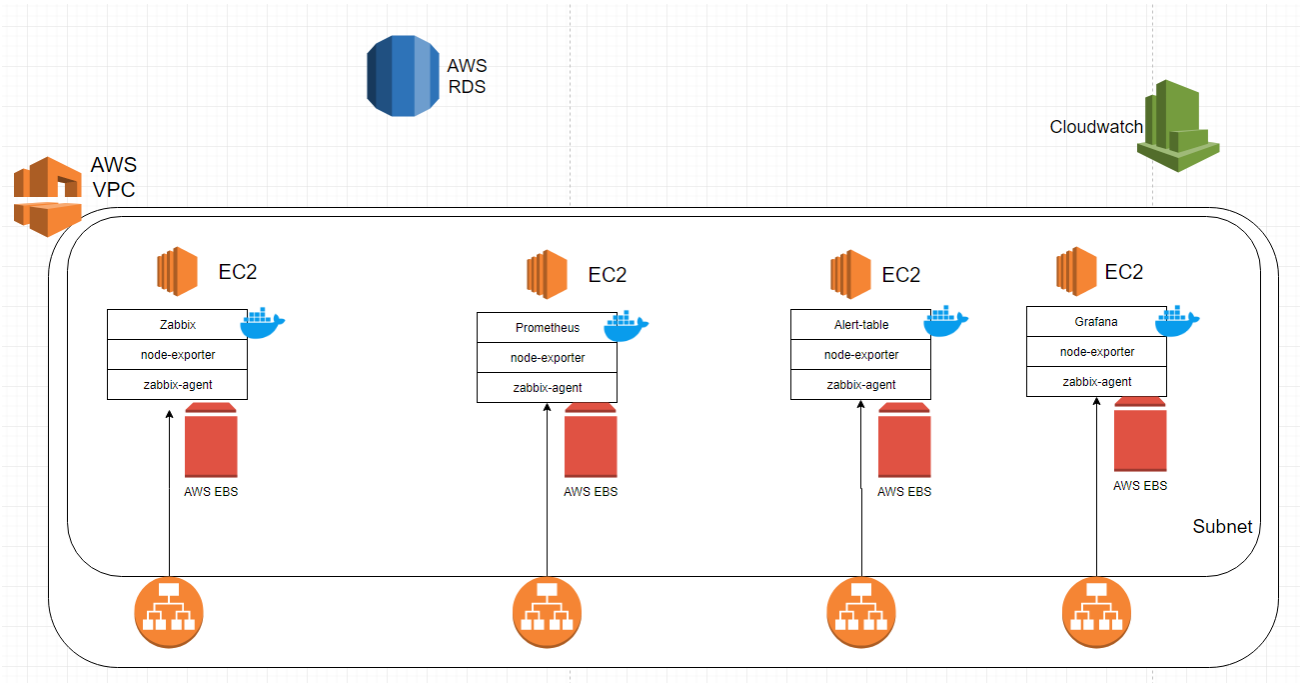

Рис.3.2. - Загальна інфраструктура системи

3.3.1.Використані компоненти.

Amazon Virtual Private Cloud (Amazon VPC) дозволяє запускати ресурси AWS у віртуальну мережу. Ця віртуальна мережа дуже нагадує традиційну мережу, з перевагами використання масштабованої інфраструктури AWS. [6]

Amazon EBS надає томи блочного рівня для використання з екземплярами EC2. Томи EBS є високодоступними та надійними дисками зберігання, які можна приєднати до будь-якого запущеного екземпляра, який знаходиться в тій же зоні доступності. Томи EBS, які прикріплені до екземпляра EC2, виставляються як томи зберігання, які знаходяться незалежно від екземпляра.

Amazon EBS рекомендується, коли дані повинні бути швидко доступними і вимагають довготривалої стійкості. Томи EBS особливо добре підходять для використання в якості основного сховища для

файлових систем, баз даних або для будь-яких додатків, які вимагають точних деталізованих оновлень і доступу до необробленого, неформатованого сховища на рівні блоків. [12]

AWS Security Groups функціонує як віртуальний брандмауер для керування вхідним і вихідним трафіком. Коли запускається інстанс у VPC, можна призначити до п'яти груп безпеки для нього. Групи безпеки діють на рівні інстанса, а не на рівні підмережі. Таким чином, кожен інстанс у підмережі у VPC може бути призначений для будь-якого набору груп безпеки. Якщо не вказати конкретну групу під час запуску, інстанс автоматично призначається групі безпеки за замовчуванням для VPC. [7]

Для кожної групи безпеки додаються правила, які контролюють вхідний трафік на інстас, і окремий набір правил, які контролюють вихідний трафік. [7]

AWS ALB. Application Load Balancer функціонує на прикладному рівні моделі взаємодії відкритих систем (OSI). Після того, як балансувальник навантаження отримає запит, він оцінює правила слухача в порядку пріоритету, щоб визначити, яке правило застосовувати, а потім вибирає ціль з цільової групи для дії правила. Можна налаштувати правила слухача для маршрутизації запитів до різних цільових груп на основі вмісту трафіку програми. Маршрутизація виконується незалежно для кожної цільової групи, навіть якщо цільова група зареєстрована з кількома цільовими групами.

Можна додавати та видаляти цілі з alb, не порушуючи загальний потік запитів до програми. Еластичне балансування навантаження масштабує балансування навантаження, оскільки трафік до програми змінюється з часом. Еластичне балансування навантаження може масштабуватися до переважної більшості навантажень автоматично.

Можна налаштувати перевірки стану здоров'я, які використовуються для контролю стану зареєстрованих цілей, так що

балансування навантаження може надсилати запити тільки до справних цілей.

Amazon Relational Database Service (Amazon RDS) - це веб-сервіс, який спрощує налаштування, керування та масштабування реляційної бази даних у хмарі. Він забезпечує економічну, змінювану потужність для стандартної реляційної бази даних і управляє загальними завданнями адміністрування баз даних.

Основним блоком Amazon RDS є екземпляр БД. Екземпляр БД - це ізольована середовище бази даних у хмарі. Екземпляр БД може містити кілька створених користувачем баз даних, і до нього можна отримати доступ за допомогою тих самих інструментів і програм, які використовуються з автономним екземпляром бази даних. Можна створювати та змінювати екземпляр БД за допомогою інтерфейсу командного рядка AWS, API Amazon RDS або консолі керування AWS.

Кожен екземпляр БД запускає движок БД. В даний час Amazon RDS підтримує двигуни MySQL, MariaDB, PostgreSQL, Oracle і Microsoft SQL Server. Кожен движок БД має свої власні підтримувані функції, і кожна версія движка БД може містити певні функції. Крім того, кожен движок БД має набір параметрів у групі параметрів БД, які контролюють поведінку баз даних, якими він керує.

Інстанси БД мають три типи: магнітний, загальний (SSD) і забезпечений IOPS (PIOPS). Вони відрізняються характеристиками продуктивності та ціною, що дозволяє пристосувати продуктивність зберігання та вартість до потреб бази даних. Кожен екземпляр БД має мінімальні та максимальні вимоги до зберігання, залежно від типу зберігання та движку бази даних, який він підтримує. Важливо мати достатню кількість запам'ятовуючих пристроїв, щоб ваші бази даних мали можливість збільшуватися, а можливості механізму БД мають місце для запису вмісту або записів журналу.

AWS Backup - це повністю керований сервіс резервного копіювання, що дозволяє легко централізувати та автоматизувати резервне копіювання даних через сервіси AWS у хмарі. Використовуючи AWS Backup, можна централізовано налаштувати політики резервного копіювання та контролювати діяльність резервного копіювання ресурсів AWS. AWS Backup автоматизує і консолідує завдання резервного копіювання, відкидаючи необхідність створення спеціальних скриптів і ручних процесів. На консолі AWS Backup можна створити політики резервного копіювання, які автоматизують розклад резервних копій та керування збереженням.

AWS Backup надає повністю кероване рішення для резервного копіювання на основі політик, що спрощує керування резервними копіями.

AWS Backup поділений на такі примітиви:

- Backup plan (толькі для автоматичного створення резервних копій)
- Vault
- Selection

У AWS Backup план резервного копіювання є об’єкт , який визначає, коли і як потрібно створювати резервні копії ресурсів AWS, таких як таблиці Amazon DynamoDB або файлові системи Amazon Elastic File System (Amazon EFS). Можна призначити ресурси для резервних планів, і AWS Backup автоматично створює резервні копії і зберігає резервні копії цих ресурсів відповідно до плану резервного копіювання.

Можна створити кілька планів резервного копіювання з різними вимогами до резервного копіювання.

Частота резервного копіювання. Частота резервного копіювання визначає частоту створення резервної копії. Можна обрати частоту кожні 12 годин, щодня, щотижня або щомісяця. При щотижневому розкладі,

можна вказати, в які дні тижня потрібно робити резервні копії. При місяцевому розкладі можна вибрати певний день місяця.

Вікна резервного копіювання складаються з часу, коли починається резервне копіювання, і тривалості у годинах. У цьому вікні запускаються резервні завдання. Якщо за якихось причин виникають труднощі з визначенням часу для вікна резервного копіювання, можна скористатися типовим резервним копіюванням, який рекомендує AWS Backup. Вікно резервного копіювання за умовчанням починається з 5 ранку UTC і триває 8 годин.

Життєвий цикл визначає, коли резервна копія переходить на холодне сховище та коли вона видаляється. AWS Backup переводить резервні копії на інший тип сховища, а потім видаляє їх у відповідності з правилами, визначеними користувачем. Резервні копії, перенесені на холодне зберігання, повинні зберігатися в там протягом мінімум 90 днів.

Резервне сховище - це контейнер для організації резервних копій. Резервні копії, створені за допомогою правила резервного копіювання, доставляються у сховище резервних копій, яке вказано в правилі резервного копіювання. Резервні копії у сховищі можна шифрувати за допомогою AWS Key Management Service (AWS KMS). Також можна додати теги до резервних сховищ, для їх легшої організації. Також можливе створення власного сховища для резервних копій, якщо стандартне за якихось причин не задовольняє вимоги користувача.

Amazon Route 53 - це високодоступна й масштабована веб-служба системи доменних імен (DNS). Route 53 можна використовувати для виконання трьох основних функцій у будь-якій комбінації: реєстрація домену, маршрутизація DNS та перевірка стану здоров'я веб-додатків. [14]

Основні терміни:

1) Доменне ім'я

2) Реєстр доменів
3) Домени вищого рівня
4) Авторитетний сервер імен
5) Запит DNS
6) DNS resolver
7) DNS
8) IP address
9) hosted zone
10) name servers
11) private DNS
12) DNS запис

Домене ім’я. Ім'я, яке користувач вводить у адресний рядок веб-браузера для доступу до веб-сайту або веб-програми. Щоб зробити свій веб-сайт або веб-додаток доступним в Інтернеті, необхідно зареєструвати домене ім’я.

Реєстр доменів. Компанія, яка має право продавати домени з певним доменом верхнього рівня. Наприклад, VeriSign - це реєстр, який має право продавати домени, які мають домен .com. Реєстр доменів визначає правила реєстрації домену, такі як вимоги до місця проживання для географічного TLD. Реєстр доменів також підтримує достовірну базу даних для всіх доменних імен, які мають один і той самий TLD. База даних реєстру містить таку інформацію, як контактна інформація та сервери імен для кожного домену.

Домени вищого рівня - остання частина доменного імені, наприклад .com, .org або .ninja. Існує два типи доменів верхнього рівня:

1) загальні домени вищого рівня - Ці TLD зазвичай дають користувачам уявлення про те, що вони знайдуть на веб-сайті. Наприклад, доменні імена, які мають TLD .bike, часто асоціюються з веб-сайтами для підприємств, які замайються велосипедами. За

кількома винятками можна скористатися будь-яким загальним TLD.

2) географічні домени вищого рівня - ці TLD пов'язані з такими географічними областями, як країни або міста. Деякі реєстри географічних TLD мають вимоги до місця проживання, інші, наприклад, .io, дозволяють або навіть заохочують використання у якості загального TLD.

Авторитетний сервер імен. Сервер імен, який має остаточну інформацію про одну частину системи доменних імен (DNS) і відповідає на запити від розпізнавача DNS, повертаючи відповідну інформацію. Наприклад, авторитетний сервер імен для домену верхнього рівня .com (TLD) знає імена серверів імен для кожного зареєстрованого домену .com. Коли авторитетний сервер імен .com отримує запит від розпізнавача DNS для example.com, він відповідає іменами серверів імен для служби DNS для домену example.com.

Сервери імен route 53 є авторитетними серверами імен для кожного домену, який використовує route 53 як службу DNS. Сервери імен знають, як потрібно спрямовувати трафік для домену та субдоменів на основі записів, створених у зоні розміщення домену.

Наприклад, якщо сервер імен route 53 отримує запит на www.example.com, він знаходит DNS запис і повертає IP-адресу, таку як 192.0.2.33, що вказано в записі.

DNS запит. Зазвичай це запит, який подається пристроєм, наприклад комп'ютером або смартфоном, до системи доменних імен (DNS) для ресурсу, пов'язаного з доменним ім'ям. Найпоширенішим прикладом запиту DNS є те, коли користувач відкриває браузер і вводить доменне ім'я в адресному рядку. Відповідь на запит DNS зазвичай - це IP-адреса, пов'язана з таким ресурсом, як веб-сервер. Пристрій, який ініціював запит, використовує IP-адресу для зв'язку з ресурсом.

Наприклад, браузер може використовувати IP-адресу для отримання веб-сторінки з веб-сервера.

DNS resolver. DNS-сервер, який часто управляється постачальником послуг Інтернету (ISP), який виступає посередником між запитами користувачів і серверами імен DNS. Коли користувач відкриває веб-браузеп і вказує доменне ім'я в адресному рядку, запит спочатку переходить до розпізнавача DNS. Розпізнавач здійснює зв'язок з серверами імен DNS, щоб отримати IP-адресу для відповідного ресурсу, наприклад, веб-сервера. Розчинювач DNS також відомий як рекурсивний сервер імен, оскільки він надсилає запити до послідовності авторитетних серверів імен DNS, поки не отримає відповідь (як правило, IP-адресу), що він повертається на пристрій користувача, наприклад, веб-браузер на ноутбук.

DNS. Всесвітня мережа серверів, які допомагають комп'ютерам, смартфонам, планшетам та іншим пристроям з підтримкою IP-зв'язку спілкуватися один з одним. Система доменних імен переводить легко зрозумілі назви, такі як example.com, у числа, відомі як IP-адреси, які дозволяють комп'ютерам знаходити один одного в Інтернеті.

Hosted zone. Контейнер для записів, у якому міститься інформація про те, як потрібно спрямовувати трафік для домену (наприклад example.com) і всіх його субдоменів (наприклад, www.example.com, retail.example.com і seattle.accounting example.com). Зона розміщення має таку ж назву, що і відповідний домен.

Наприклад, розміщена зона example.com може містити запис, що містить інформацію про маршрутизацію трафіку для www.example.com на веб-сервер, який має IP-адресу 192.0.2.243, і запис, що містить інформацію про маршрутизацію електронної пошти. .com до двох серверів електронної пошти, mail1.example.com і mail2.example.com. Кожен поштовий сервер також потребує власного запису.

IP адреса. Номер, призначений пристрою в Інтернеті, наприклад, ноутбуку, смартфону або веб-серверу, який дозволяє пристрою спілкуватися з іншими пристроями в Інтернеті. Наданий момент існує два типи IP адрес:

1) IPv4 – використовує 32 бітні адреси. Максимальна кількість адрес - 4 294 967 296. Адреси мають формат: 192.168.1.1 [1]
2) IPv6 – використовує 128 бітні адреси, через що значно виросла загальна кількість адрес. [1]

Route 53 підтримує IPv4 та IPv6, тобто дозволяється створювати A записи для IPv4 і AAAA записи для IPv6, налаштовувати перевірки стану веб-служб тощо.

DNS запис. Об'єкт у розміщеній зоні, який використовується для визначення шляху маршрутизації трафіку для домену або субдомену. Наприклад, можна створити записи для example.com і www.example.com, які спрямовують трафік до веб-сервера, який має IP-адресу 192.0.2.234.

Сервер імен. Сервери в системі доменних імен (DNS), які допомагають переводити доменні імена в IP-адреси, які комп'ютери використовують для спілкування один з одним. Серверів імен є або рекурсивними серверами імен (також відомими як DNS resolver) або авторитетними серверами імен.

Приватний сервер імен. Локальна версія системи доменних імен (DNS), яка дозволяє направляти трафік для домену та його субдоменів до екземплярів Amazon EC2 в межах одного або декількох віртуальних приватних Amazon (VPC).

3.4. Розгортання локально.

Для локального розгортання інфраструктури с усіма вище описаними система моніторингу, пропонується використовувати docker-compose. Це утиліта, що дозволяє описувати та запускати декілька

контейнерів. Такий спосіб забезпечує відмовостійкості, високої доступность тощо, проте може зручним для локальної розробки та тестування.

3.5. Висновки до розділу 3

У третьому розділі бакалаврської роботи виконано дослідження методів розгортання розробленого веб-серверу та використаних систем моніторингу. У результаті можна зробити висновки:

1) розглянуто основні принципи побудови надійних систем, такі як: відмовостійкість, надлишковість, безпека, створення та зберігання резервних копій. Розглянуто сучасний метод розгортання програмного забезпечення – контейнеризація, з використанням технології docker.
2) запропонована інфраструктура системи на базі хмарного провайдеру AWS. Розглянуті основні сервіси даного хмарного провайдеру, які пропонуються до використання проведений опис їх переваг та недоліків, побудована загальна схема інфраструктури.
3) запропонований метод локального розгортання системи, призначений для подальшої розробки та тестування системи візуалізації критичних повідомлень різних засобів моніторингу.

ВИСНОВКИ

Під час виконання бакалаврської роботи було виконано розробку централізованої системи візуалізації критичних повідомлень різних систем моніторингу, яка інтегрується з системами моніторингу та є загальної точкою для отримання інформації про критичні повідомлення, які виникли у системі.

У дослідженні використовуються такі системи моніторингу, як Prometheus, Zabbix, Cloudwatch, досліджуються принципи їх побудови, робиться загальний огляд їх переваг та недоліків, розглядаються особливості їх розгортання. Також досліджуються сервіси хмарного провайдеру AWS, інструменти для контейнеризації та локального розгортання такі як, docker та docker-compose.

Метою роботи було зібрати активні критичні повідомлення та візуалізувати, тобто надати єдиний інтерфейс, на якому можна переглянути стан системи в цілому.

Для вирішення поставленої задачі був розроблений веб-сервер, який інтегрується з різними системами моніторингу, збирає інформацію про наявні критичні повідомлення у системі. Була запропонована інфраструктура системи, яка містить у собі декілька систем моніторингу, запропоновані методи реалізації важливих принципів побудови систем, таких як відмовістійкість, безпека, надлишковість, створення резервних копій тощо. В цілому, було запропоновано два варіанти для розгортання системи – один з використанням хмарного провайдеру та з реалізацією вищевказаних принципів, інший – для локального розгортання в цілях подальшої розробки та тестування програмного забезпечення.

ПЕРЕЛІК ПОСИЛАНЬ

1. Таненбаум Э. Компьютерные сети / Э. Таненбаум // СПб: Питер. – 2018. – 960 с.
2. DevOps Handbook / Gene Kim , Patrick Debois, John Willis, Jez Humble, John Allspaw
3. Docker: Up & Running, 2nd Edition / Karl Matthias, Sean P. Kane
4. Docker in Action / Jeff Nickoloff
5. High Availability: Design, Techniques, and Processes / Prentice Hall // 2009
6. Amazon Web Services in Action / Andreas Wittig,Michael Wittig // 2015
7. AWS: Amazon Web Services Tutorial The Ultimate Beginners Guide / Dennis Hutten // 2017
8. Mike's Guides to Learning Boto3 / Mike Kane
9. Mastering Zabbix / Andrea Dalle Vacche // 2013
10. Prometheus: Up & Running / Brian Brazil // 2018
11. Cloudwach documentation [Електронний ресурс] // - Режим доступа: https://docs.aws.amazon.com/en_us/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html
12. EBS documentation [Електронний ресурс] // - Режим доступа: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.htm
13. SNS documentation [Електронний ресурс] // - Режим доступа: https://docs.aws.amazon.com/en_us/sns/latest/dg/welcome.html
14. Route 53 documentation [Електронний ресурс] // - Режим доступа: https://docs.aws.amazon.com/en_us/Route53/latest/DeveloperGuide/Welcome.html

15. Boto3 documentation [Електронний ресурс] // - Режим доступа: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html?id=docs_gateway

ДОДАТКИ

Дадаток А

Main.py

```
#!/usr/bin/python3


import cloudwatch
import prometheus
from flask import Flask, request, jsonify
import argparse


app = Flask(__name__)

import argparse
parser = argparse.ArgumentParser()
parser.add_argument("prometheus", help="prometheus_host")
args = parser.parse_args()
client = cloudwatch.CloudwatchClient()
client_prom = prometheus.PrometheusClient(args.prometheus)


@app.route("/")
def health_check():
    return "The alert-table datasource is healthy."


@app.route("/search", methods=["POST"])
```

```
def search():
    return jsonify(
        [
            "cloudwatch_green",
            "cloudwatch_red",
            "cloudwatch_insufficient_data",
            "prometheus_red",
            "prometheus_all",
        ]
    )


@app.route("/query", methods=["POST"])
def query():
    if request.get_json()["targets"][0]["target"] == "cloudwatch_red":
        alarms = client.getAlarm()
        return jsonify(alarms)
    elif request.get_json()["targets"][0]["target"] ==
"cloudwatch_insufficient_data":
        alarms = client.getAlarm(alarm_state="INSUFFICIENT_DATA")
        return jsonify(alarms)
    elif request.get_json()["targets"][0]["target"] == "cloudwatch_green":
        alarms = client.getAlarm(alarm_state="OK")
        return jsonify(alarms)
    elif request.get_json()["targets"][0]["target"] == "prometheus_red":
        alerts = client_prom.getAlerts()
        return jsonify(alerts)
    elif request.get_json()["targets"][0]["target"] == "prometheus_all":
        alerts = client_prom.getRules()
```

```
        return jsonify(alerts)


if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

cloudwatch.py

```
import boto3
from json import loads


class CloudwatchClient:
    def __init__(self):
        self.client = boto3.client("cloudwatch")

    def getAlarm(self, alarm_state="ALARM"):
        alarm_list = []
        alarm_list_json = []
        response = self.client.describe_alarms(StateValue=alarm_state)
        alarm_list_json += response["MetricAlarms"]
        while response.get("NextToken", False):
            response = self.client.describe_alarms(
                StateValue=alarm_state, NextToken=response["NextToken"]
            )
            alarm_list_json += response["MetricAlarms"]
        for alarm in alarm_list_json:
            alarm_list.append(
                dict(
```

```
                {
                    "Alarm": alarm["AlarmName"],
                    "StateReason": alarm["StateReason"],
                    "MetricName": alarm["MetricName"],
                    "Threshold": alarm["Threshold"],
                }
            )
        )
    return self.table_cloudwatch(alarm_list)

def table_cloudwatch(self, alarms):
    rows = []
    for alarm in alarms:
        rows.append(
            [
                alarm["Alarm"],
                alarm["StateReason"],
                alarm["MetricName"],
                alarm["Threshold"],
            ]
        )
    data = [
        {
            "columns": [
                {"text": "Alert", "type": "string"},
                {"text": "StateReason", "type": "string"},
                {"text": "MetricName", "type": "string"},
                {"text": "Threshold", "type": "string"},
            ],
```

```
            "rows": rows,
            "type": "table",
        }
    ]
    return data
```

prometheus.py

```
import requests

class PrometheusClient:
    def __init__(self, prometheus_url):
        self.prometheus_url = prometheus_url

    def getAlerts(self):
        prometheus_response = requests.get(
            "http://{}/api/v1/alerts".format(self.prometheus_url)
        ).json()
        alert_list = []
        for alert in prometheus_response["data"]["alerts"]:
            alert_list.append(
                dict(
                    {
                        "Alert": alert["labels"]["alertname"],
                        "Description": alert["annotations"]["description"],
                        "State": alert["state"],
                        "Value": alert["value"],
                    }
                )
```

```
            )
        return self.table_alerts(alert_list)

    def getRules(self):
        prometheus_response = requests.get(
            "http://{}/api/v1/rules".format(self.prometheus_url)
        ).json()
        alert_list = []
        for rules_set in prometheus_response["data"]["groups"]:
            for rule in rules_set["rules"]:
                alert_list.append(
                    dict(
                        {
                            "AlertName": rule["name"],
                            "PrometheusQuery": rule["query"],
                            "Duration": rule["duration"],
                        }
                    )
                )
        return self.table_rules(alert_list)

    def table_alerts(self, alerts):
        rows = []
        for alert in alerts:
            rows.append(
                [alert["Alert"], alert["Description"], alert["State"],
alert["Value"]]
            )
        data = [
```

```
            {
                "columns": [
                    {"text": "Alert", "type": "string"},
                    {"text": "Description", "type": "string"},
                    {"text": "State", "type": "string"},
                    {"text": "Value", "type": "string"},
                ],
                "rows": rows,
                "type": "table",
            }
        ]
        return data
    def table_rules(self, rules):
        rows = []
        for rule in rules:
            rows.append([rule["AlertName"], rule["PrometheusQuery"],
rule["Duration"]])
        data = [
            {
                "columns": [
                    {"text": "AlertName", "type": "string"},
                    {"text": "PrometheusQuery", "type": "string"},
                    {"text": "Duration", "type": "string"},
                ],
                "rows": rows,
                "type": "table",
            }
        ]
        return data
```